MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21b/2014/07.28.19.09-TDI

PROJETO E DESENVOLVIMENTO DE UM CONTROLADOR DE MOTORES "BRUSHLESS" (BLDC) PARA APLICAÇÃO EM VOLANTES DE INÉRCIA

Fernando de Almeida Martins

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Mecânica Espacial e Controle, orientada pelo Dr. Valdemir Carrara, aprovada em 29 de maio de 2014.

URL do documento original:
<http://urlib.net/8JMKD3MGP5W34M/3GNNJ2B>

INPE
São José dos Campos
2014

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/6921
Fax: (012) 3208-6919
E-mail: pubtc@sid.inpe.br

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

sid.inpe.br/mtc-m21b/2014/07.28.19.09-TDI

PROJETO E DESENVOLVIMENTO DE UM CONTROLADOR DE MOTORES "BRUSHLESS" (BLDC) PARA APLICAÇÃO EM VOLANTES DE INÉRCIA

Fernando de Almeida Martins

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Tecnologia Espaciais/Mecânica Espacial e Controle, orientada pelo Dr. Valdemir Carrara, aprovada em 29 de maio de 2014.

URL do documento original:
<http://urlib.net/8JMKD3MGP5W34M/3GNNJ2B>

INPE
São José dos Campos
2014

Dados Internacionais de Catalogação na Publicação (CIP)

Martins, Fernando de Almeida.

M366p Projeto e desenvolvimento de um controlador de motores "brushless" (BLDC) para aplicação em volantes de inércia / Fernando de Almeida Martins. – São José dos Campos : INPE, 2014.

xxii + 119 p. ; (sid.inpe.br/mtc-m21b/2014/07.28.19.09-TDI)

Dissertação (Mestrado em Engenharia e Tecnologia Espaciais/Mecânica Espacial e Controle) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2014.

Orientador : Dr. Valdemir Carrara.

1. Roda de reação. 2. Controlador eletrônico. 3. Motor sem escovas BLDC. 4. FPGA. 5. Satélite. I.Título.

CDU 629.7.062.2

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de ***Mestre*** em

Engenharia e Tecnologia Espaciais/Mecânica Espacial e Controle

Dr. Hélio Koiti Kuga

Presidente / INPE / SJCampos - SP

Dr. Valdemir Carrara

Orientador(a) / INPE / SJCampos - SP

Dr. Mario Cesar Ricci

Membro da Banca / INPE / SJCampos - SP

Dr. Paulo Giácomo Milani

Membro da Banca / INPE / SJCampos - SP

Dr. Fabricio de Novaes Kucinskis

Membro da Banca / INPE / São José dos Campos - SP

Dr. Rodrigo Alvite Romano

Convidado(a) / IMT / São Caetano do Sul - SP

Este trabalho foi aprovado por:

( ) maioria simples

(X) unanimidade

Aluno (a): ***Fernando de Almeida Martins***

São José dos Campos, 29 de Maio de 2014

“Estudar é um ato político”.

F. Martins

Aos que ainda acreditam que a educação é a força máxima de um país.

AGRADECIMENTOS

Agradeço aos ensinamentos dos mestres do Instituto Nacional de Pesquisas Espaciais que abriram meus horizontes para a pesquisa no ramo da engenharia e tecnologias espaciais, em especial ao Dr. Valdemir Carrara pelo empenho em orientar este trabalho. Agradeço às contribuições do Dr. José Carlos de Souza Junior, do Dr. Rodrigo Alvite Romano e do Dr. Vanderlei Cunha Parro do Instituto Mauá de Tecnologia, pelas inestimáveis informações prestadas e pelo constante apoio e incentivo ao meu desenvolvimento profissional e acadêmico. Por fim e não menos importante, agradeço a Deus por estar presente em minha vida, agradeço a participação da esposa pelo companheirismo e amor e agradeço também a família e amigos pelo estímulo ao meu crescimento como ser humano, na paciência pelo tempo investido e no apoio recebido perante as dificuldades da vida.

RESUMO

Este trabalho apresenta o projeto e o desenvolvimento de uma eletrônica para controle de motores de corrente contínua sem escovas (*Brushless DC Motor* - BLDC). A aplicação visada é o emprego deste controlador em rodas de reação para sistemas de controle de atitude de satélites. O principal objetivo e o foco da aplicação foi o desenvolvimento e construção de vários métodos para acionar o motor e medir os parâmetros de desempenho. A placa projetada criou uma plataforma adequada para o estudo do acionamento, e testes de software para o acionamento das fases do BLDC. O uso de dispositivos programáveis, como micro-controladores e dispositivos programáveis lógicos como FPGA permitiram projetar diferentes estratégias de controle. Combinando vários tipos de sensores e dados externos adquiridos de sensores pela placa eletrônica, procedimentos de controle em malha fechada puderam ser experimentados com diferentes sensores. Os resultados são apresentados e o desempenho é comparado com o de uma roda de reação típica.

DESIGN AND CONSTRUCTION OF A MOTOR CONTROLLER "BRUSHLESS" (BLDC) APPLICATION FOR STEERING WHEELS AND WHEELS OF INERTIA

ABSTRACT

This work presents the design and development of an electronic board to control Brushless DC Motors (BLDC). The main application of this work is to use the board to control the BLDC of a reaction wheel and to apply it to satellite attitude control systems (ACS). The focus was the development and construction of several methods to drive the motor and to measure the performance parameters. The designed board created a plataform suitable for BLDC driving studies and software testing. The use of programmable devices like micro-controllers and logical programmable devices like FPGA allows to project different control strategies. Combining many types of sensors and external data acquired by the electronic board, closed loop control procedures could be experimented with different sensors. The results collected from control performance and a comparison of motor performance with a typical RW were presented.

LISTA DE FIGURAS

LISTA DE SIGLAS E ABREVIATURAS

INPE	Instituto Nacional de Pesquisas Espaciais
SID	Serviço de Informação e Documentação
TDI	Teses e Dissertações Internas
SPG	Serviço de Pós-Graduação
FPGA	Field-Programmable Gate Array
BLDC	Brushless Direct Current
VHDL	VHSIC Hardware Description Language
VHSIC	Very-High-Speed Integrated Circuit
SCD	Satélite de Coleta de Dados
PWM	Pulse Width Modulation
ADC	Analog to Digital Converter
LCD	Liquid Cristal Display
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
ASIC	Application Specific Integrated Circuit
CC	Corrente Contínua
DMA	Direct Memory Access
EMC	Electromagnetic Compatibility
LC	Circuito paralelo de uma bobina (L) e um capacitor (C)
PID	Controlador Proporcional Integral e Derivativo
RPM	Rotações Por Minuto

SUMÁRIO

1. INTRODUÇÃO

Os satélites artificiais normalmente giram ao redor da Terra, embora também possam ser colocados em órbita ao redor da Lua, do Sol ou de outros planetas. O movimento orbital pode ser entendido como o movimento de um ponto de massa ao redor da Terra e este ponto representa toda a massa do satélite (SOUZA, 1987), que se mantém em órbita devido à aceleração da gravidade e à sua velocidade. Dessa maneira, ele permanece em constante queda livre em torno da Terra, comportando-se como se estivesse “preso” em sua órbita.

Ao longo do tempo, outras forças atuam sobre o satélite, como o torque aerodinâmico (produzido pela interação da superfície da espaçonave com a atmosfera superior), o torque de gradiente de gravidade (causado em objetos não simétricos devido à variação da força gravitacional da Terra em satélites com assimetria de massa), o torque magnético (resultado da interação de campos magnéticos residuais da espaçonave com o campo magnético terrestre) e o torque de radiação solar (devido à radiação solar que incide na superfície da espaçonave).

Esses torques perturbam a órbita dos satélites e modificam os elementos orbitais. São efeitos pequenos, mas que somados ao longo do tempo causam alterações no movimento orbital. Por isto, os satélites precisam ser equipados com dispositivos para corrigir sua órbita, , como os jatos de gás ou de hidrazina.

Os satélites apresentam também uma dinâmica em torno do seu centro de massa. Esse movimento define o movimento de atitude, ou seja, a orientação que o satélite assume no espaço.

Em um satélite em órbita é comum que este sofra a ação dos torques ambientais externos constantemente, como mencionado anteriormente, que modificam a atitude de forma indesejada, criando assim a necessidade de correções, que podem ser feitas por meio de atuadores, como uma roda de reação, por exemplo.

A roda de reação usa o princípio da conservação da quantidade de momento angular que diz que em um sistema livre de torques externos a quantidade de momento angular se conserva. De uma forma simplória, pode-se dizer que rodas de reação são motores elétricos CC dotados de volantes de inércia.

Quando o motor imprime uma velocidade de rotação ao volante de inércia, o satélite, que é solidário a roda, gira em sentido oposto. Dessa forma é possível corrigir a atitude do satélite (FONSECA, 2011). A roda de reação, ao contrário dos jatos de gás e de bobinas geradoras de torque magnético (magnetotorques), geram torques internos.

Para que se tenha uma correção na atitude nos 3 eixos do satélite normalmente utilizam-se 3 rodas de reação em um arranjo tri-ortogonal com uma quarta roda redundante com seu eixo de rotação na bissetriz do triedro formado pelas 3 outras.

1.1. Objetivo

Este trabalho tem por objetivo projetar e construir um protótipo de uma roda de reação com características funcionais semelhantes àquelas voltadas para aplicações espaciais. A roda desenvolvida é capaz de realizar os acionamentos elétricos de fase para motores de corrente contínua sem escovas (ou BLDC, do Inglês Brushless DC motor) em diversas configurações de polos e fases para se criar os fundamentos requeridos pela eletrônica de uma roda de reação.

O circuito de acionamento eletrônico leva em conta a necessidade de estudar os diversos métodos para se energizar um motor BLDC e as muitas maneiras e métodos de construí-los, com a intenção de servir como um modelo experimental, onde novas técnicas de controle e métodos de acionamento possam ser geradas e testadas. Assim, a placa eletrônica para controle é capaz de permitir alterar o modo de acionamento das fases por software, possibilitando diversos ensaios em uma mesma plataforma.

Em resumo, este trabalho apresenta o projeto, desenvolvimento, implementação e testes de um dispositivo eletrônico composto de um micro-controlador e uma unidade de FPGA para o controle e acionamento das fases do motor BLDC, com capacidade de sensoriamento por meio de sensores *Hall*, por sensor óptico (*encoder)*, por medição da corrente das fases e medição das tensões de cada fase. Além da alimentação do motor, foi implementado o monitoramento remoto e a medição da rotação e da corrente do motor, de modo a permitir que se controle o funcionamento do motor numa aplicação típica para uma roda de reação em malha fechada.

1.2. Motivação

Este trabalho tem como motivação a necessidade do país de dominar a tecnologia de rodas de reação voltadas para aplicação espacial. A área de aplicações espaciais e de desenvolvimento de satélites artificiais deixou de ser um desafio e passou a ser uma realidade essencial ao desenvolvimento desta nação. A vida atual está repleta de atividades cotidianas que utiliza, de alguma forma, recursos tecnológicos espaciais. As telecomunicações, a meteorologia, as pesquisas de recursos naturais por meio de sensoriamento remoto, o monitoramento de atividades e sistemas terrestres (agricultura, mineração, meio ambiente etc.), o sistema de posicionamento global para navegação de navios, aviões e carros, envolvem a utilização de satélites artificiais (SANTANA, 2008).

O custo de uma imagem de satélite meteorológico, ou da utilização de um canal de satélite de comunicação, representa um montante de investimento relevante e que impulsiona setores importantes da economia mundial, como o de cominicação e o de navegação. Além do grande significado econômico, o estudo de sistemas espaciais oferece desafios interessantes para as engenharias, para a ciência da computação, para a ciência dos materiais, e para muitas outras áreas (SANTANA, 2008).

A dinâmica de um satélite é atitude e sua órbita. A atitude compreende a posição e velocidade angulares do satélite em torno de seu centro de massa. A órbita compreende a posição e velocidade do satélite em torno da Terra. A atitude pode ser afetada por diversos torques que têm origem no meio ambiente espacial ou gerados internamente pelo próprio satélite. A determinação da atitude é o processo de computar a orientação do satélite em relação a um sistema de referência (inercial ou não inercial). Como exemplo pode-se citar um sistema de referência fixado a algum corpo de interesse, tal como a Terra. Isso normalmente envolve diversos tipos de sensores em satélites e sofisticados procedimentos de processamento de dados (WERTZ, 1978). O controle da atitude do satélite é crucial para o adequado desempenho das suas funções que podem ser sensoriamento remoto, meteorologia e comunicação, entre outras aplicações (WERTZ, 1978).

A determinação da atitude é essencial a uma missão espacial, uma vez que esta fornece a informação da orientação em relação a um sistema de referência com o qual a dinâmica do satélite é relacionado. As rodas de reação são dispositivos que permitem atuar no controle da atitude de satélites e baseiam-se em princípios físicos básicos tais como conversão de energia elétrica em energia mecânica e conservação de momento angular.

Embora circuitos eletrônicos microprocessados para controle de motores CC (corrente contínua) ou DC (do inglês *Direct Current*) estejam disponíveis no mercado, em geral estes visam aplicações nas quais a variação na velocidade de rotação é pequena. Uma roda de reação, ao contrário, necessita operar tanto em baixas quanto em altas rotações e a estabilidade do controle na partida e parada do motor é crítica ao desempenho da roda de reação, principalmente quando se considera o torque pequeno do motor utilizado para este trabalho, cerca de 13,2 mNm/A.

O controle do apontamento de satélites artificiais exige a presença de atuadores que consigam prover torques extremamente baixos de modo a compensar as mínimas perturbações encontradas no espaço, sem que, com

isso, comprometam a estabilidade do movimento. É bastante comum o emprego de rodas de reação para esta finalidade, pois elas conseguem suprir torques numa faixa de 10^{-5} a 10^{-1} Nm.

Rodas de reação são dispositivos compostos por um motor CC sem escovas (Brushless DC Motor – BLDC) acoplado a um rotor de alta inércia, quando comparado ao torque do motor, e uma eletrônica para controlar a corrente e, não simultaneamente, também a velocidade de rotação do rotor de inércia. Essa eletrônica mencionada é justamente o que este trabalho pretende estudar.

Atualmente, a maioria dos satélites que necessitam de um apontamento preciso, como os telescópios espaciais, satélites de comunicação, satélites científicos e de observação da Terra, costumam empregar rodas de reação no sistema de controle.

O desenvolvimento de rodas de reação no Brasil teve início já na década de 80 com trabalhos de mestrado cobrindo aspectos do projeto do mancal (SOUZA, 1994 e 1995), controle (TRIVELATO, 1988; TRIVELATO; SOUZA, 1988) e culminando no projeto de uma roda experimental (SOUZA; FLEURY, 1987), que foi embarcada no satélite SDC-2 lançado em 1998 e ainda em operação.

Tal trabalho envolveu diversos campos de atuação e extensos trabalhos tais como a escolha do modelo com base na literatura, especificação do equipamento, projeto mecânico, eletrônico e de software, testes funcionais e de desenvolvimento, manufatura eletrônica, sua integração e testes funcionais, desenvolvimento do software e o gerenciamento das atividades de manufatura, mecânica, integração e testes ambientais do equipamento montado (que pode ser visto na Fig. 1.1). Assim observa-se a extrema importância do estudo de acionadores de rodas de reação.

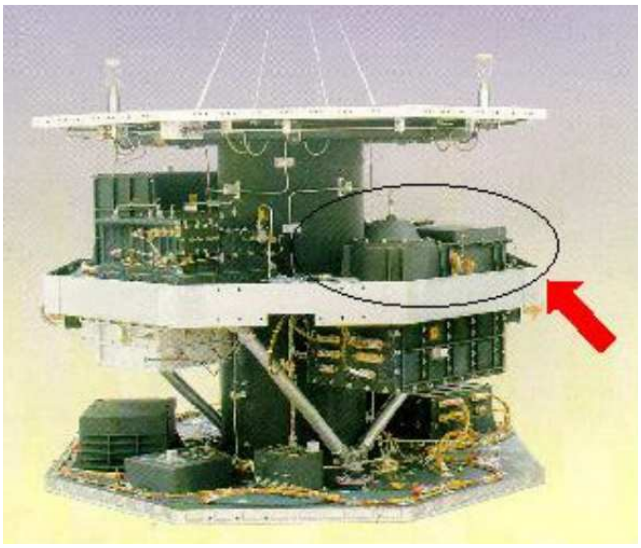


Figura 1.1 - Foto do SCD-2 com a Roda de Reação indicada no detalhe.

1.3. Metodologia

Para a realização deste projeto será necessário projetar um circuito eletrônico capaz de acionar um motor sem escovas de 3 fases com controle de velocidade de rotação e controle de corrente. Será utilizada uma configuração de chaves MOSFETS (do inglês *Metal Oxide Semiconductor Field Effect Transistor* ou transistor de efeito de campo com semicondutor em óxido metálico) em ponte H para as 3 fases efetuando o acionamento das bobinas motoras juntamente com um sistema de sensores de efeito Hall magnéticos defasados de 120° que permitirá posicionar o correto instante de chaveamento das fases.

O acionamento do motor será realizado por um microcontrolador, cuja representação simplificada é vista na Figura 1.2. O torque é controlado indiretamente por meio da corrente nas fases do motor e esta é ajustada por um acionador PWM (do inglês *Pulse Width Modulation* ou modulação por largura de pulso), com base na tensão sobre um resistor *shunt* de 0,1ohms.

Pretende-se, após realizar o projeto, validar o circuito de acionamento dos MOSFET, e efetuar-se a implementação de um controlador PI (proporcional-integral) para operar o motor em malha fechada, tanto em termos de controle de corrente como controle de velocidade angular. A sintonia do controlador

deverá ser feita manualmente segundo as regras básicas de sintonia (ÅSTRÖM,1995).

Embora pretenda-se implementar o controle eletrônico em um micro-controlador, será elaborada uma arquitetura para a placa controladora na qual pode-se tanto trabalhar com o micro-controlador quanto com uma FPGA. Esta FPGA fará a interface comos sinais de controle das fases do motor e os sinais de retorno do sistema, como o valor da corrente do motor, por exemplo, os sinais dos sensores de efeito Hall e os sinais para o codificador ótico (*encoder*). Na Fig. 1.2 tem-se o diagrama de blocos do planejado para o controlador eletrônico.

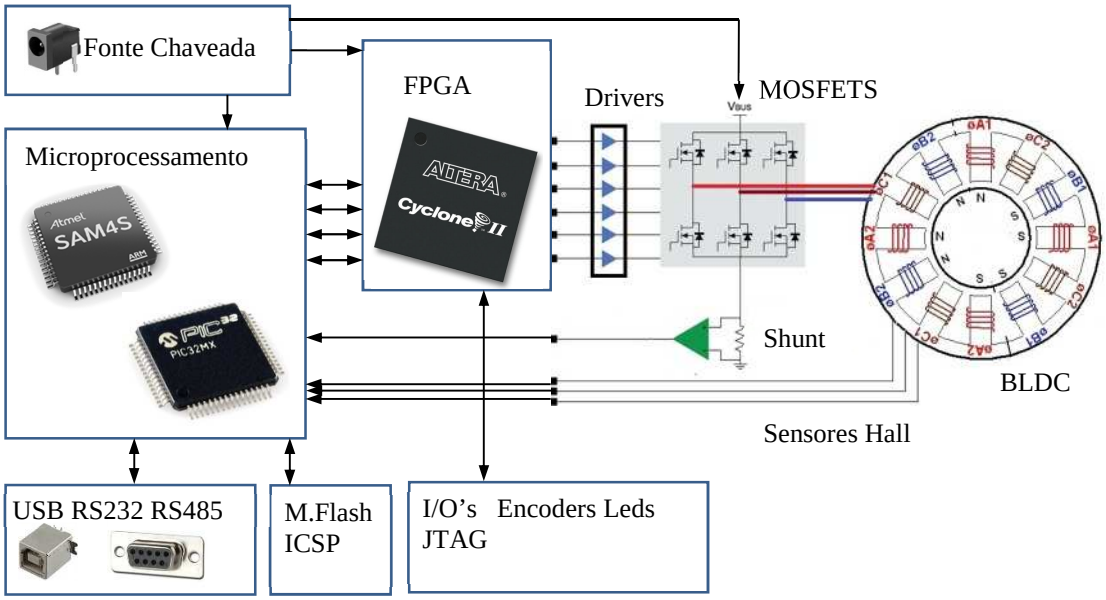


Figura 1.2 Diagrama em blocos do controlador eletrônico

A seguir, será acoplado ao motor um volante de inércia de 1,77 x 10^{-3} Kgm², e uma nova sintonia do controlador será realizada, deixando o equipamento apto para os testes em resposta ao degrau e estimação de parâmetros.

Os ensaios realizados no conjunto são relatados no capítulo 5.

2. TEORIA DE CONTROLE DO MOTOR BLDC

Para o trabalho proposto, o motor BLDC a ser utilizado será um motor DC sem escovas convencional acoplado a um volante de inércia com características semelhantes às de uma roda de reação comercial de fabricação da SunSpace, isto é, momento de inercia próximo a 1.5×10^{-3} $kg.m^2$, rotação entre -4000 e 4000rpm e torque máximo de 50 mNm. O motor utilizado será composto por 4 pares de polos e 3 fases, com uma tensão de alimentação de 12 Volts DC e uma corrente máxima nominal contínua de 1000mA. Segundo o manual do fabricante, a inércia do rotor é de 13,9 g cm² e será levada em consideração no cálculo da massa de inércia total. Além dessas propriedades o motor também disponibiliza 3 sensores de efeito Hall para o sincronismo no chaveamento das fases, acoplados ao próprio motor.

Um dos métodos mais simples de comutação do motor sem escovas de 3 fases é o método de comutação de 6 passos. Neste método, cada fase de tensão é ativada a cada 15 graus. Isto pode ser realizado pela configuração do comutador mostrado na Fig. 2.1.

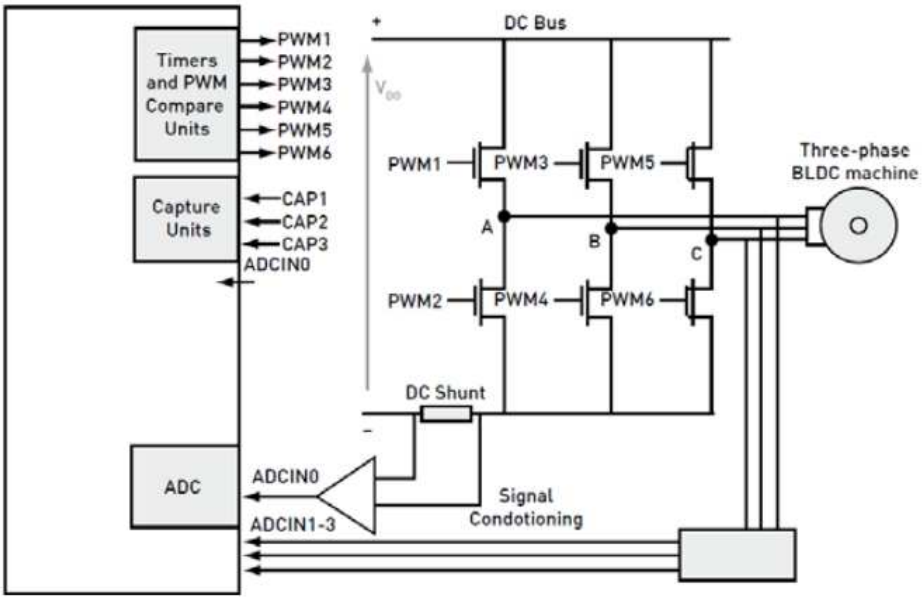


Figura 2.1. - Circuito micro-processado para acionamento dos *drivers* MOSFETS.

Os intervalos de comutações, mostrados na Fig. 2.2(b) estão em função do ângulo de rotação do rotor, cuja comutação é apresentada na Fig. 2.2(a), que representa as lógicas obtidas pelas leituras dos sensores Halls . Na Fig 2.2(c) tem-se a forma de onda completa nas bobinas do motor de 3 fases.

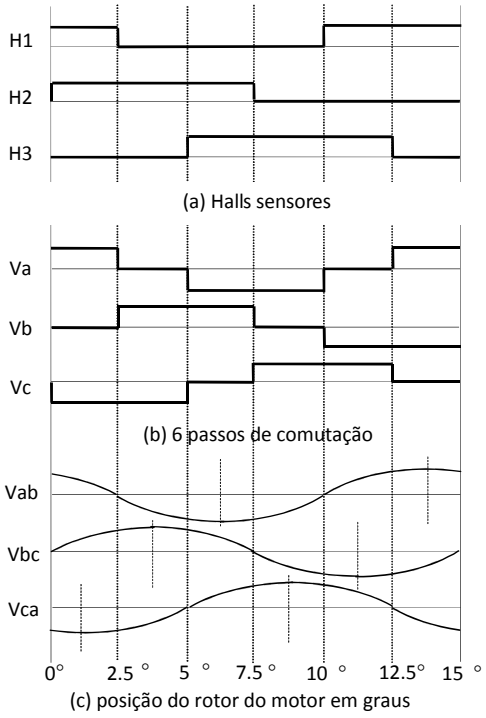


Figura 2.2 - Diagrama de fases do sensor Hall e acionamento dos MOSFETS.

Cada tensão de fase é ativada por um MOSFET superior ligado em conjunto com outro MOSFET inferior, desde que não sejam de mesmo índice para evitar uma corrente de curto-circuito entre Vcc e a linha de retorno (terra). Em outras palavras, cada ramo da ponte H trifásica composta por transistores MOSFET pode atuar em um dos três estados: positivo, negativo, ou flutuante.

Num dado instante, um polo de cada uma das fases é alimentado com tensão positiva de um lado e o polo oposto é alimentado com tensão negativa (ou terra), a fim de manter no circuito uma corrente elétrica circulando pela bobina do motor (fase). Para comutar-se apropriadamente, o controlador tem de conhecer a posição do sector (intervalo de 15 graus) do ângulo do eixo. As três

saídas do sensor Hall, como mostradas na Fig. 2.2(a), são frequentemente usadas para detectar a posição do eixo.

Este trabalho irá utilizar um motor sem escovas já com os sensores Hall embutidos no próprio motor, tendo o micro-controlador como o componente responsável pelo monitoramento da corrente e da rotação, pelo controle do sinal PWM via um controlador PI (Proporcional, Integral) e pelo sincronismo na comutação das fases, bem como por disponibilizar os dados por intermédio de uma interface serial RS232 ou USB.

Uma das partes que requer mais atenção no desenvolvimento deste acionador será a concepção de um algoritmo que garanta a operação correta nos acionamentos dos transistores MOSFETS (*drivers*) em configuração de ponte H. Isto é necessário para se evitar que uma chave conectada à alimentação (*high side*) seja acionada junto com a chave conectada ao terminal de terra *(low side*), causando assim um curto-circuito entre a alimentação da tensão e a linha de terra. No circuito de acinamento foi criado um circuito auxiliar para que este evento não ocorra e será descrito no capítulo 3.4.

Aplicando-se uma tensão controlada por PWM apenas nos transistores MOSFET superiores (H1, H2 e H3), um caminho de corrente de curto-circuito é estabelecida através de um dos diodos dos MOSFETS superiores durante o estado desligado do PWM. Por exemplo, se H1 e L2 estiverem ligados para a comutação, uma corrente é estabelecida em uma das fases do motor.

Como H1 estará gerando o sinal de PWM, então no momento que H1 for desligado, uma corrente no mesmo sentido é gerada devido ao indutor do motor estar energizado e esta corrente é forçada a circular pelo diodo acoplado ao MOSFET L1, evitando picos de tensão sofre o mosfet danificando-o. Embora L1 não necessecite ser acionado para permitir a passagem desta corrente, já que ela circula pelo diodo interno do próprio L1, esse acionamento de L1 auxilia no fluxo da corrente de descarga do indutor. A forma de onda de corrente idealizada neste momento é mostrada na Fig 2.3.

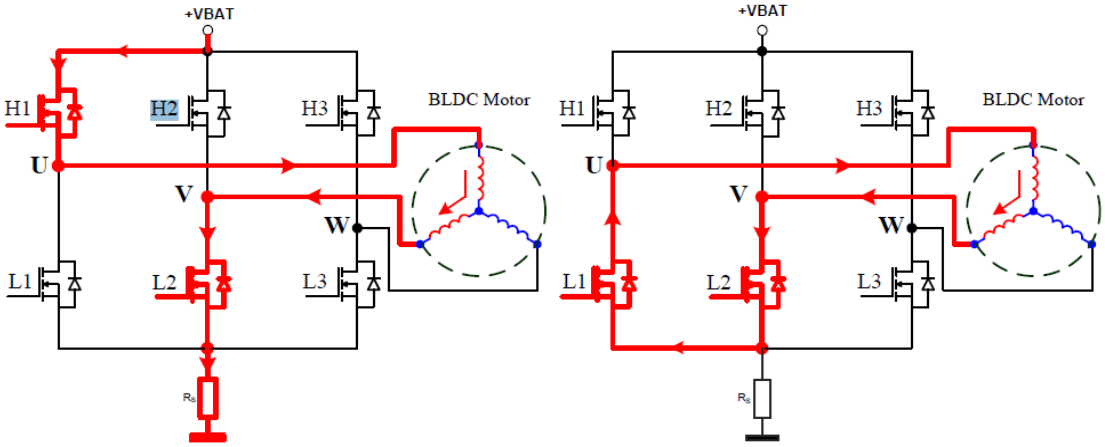


Figura 2.3. - Exemplo do fluxo de corrente no acionamento (esquerda) e quando o transistor superior é desligado (direita).

O sinal para sincronizar o chaveamento dos MOSFETS com a posição do rotor é fornecido pelos sensores de efeito *Hall* inclusos no motor. O micro controlador recebe estes pulsos em pinos de interrupção externa e computa cada posição de modo a detectar o sentido de rotação e a velocidade de rotação. Essas medidas permitirão o controle da velocidade e sentido da rotação em malha fechada. A Fig. 2.4 ilustra o processo de aquisição do sinal dos sensores *Hall*, bem como o sistema de acionamento e comutação das fases pela ponte H.

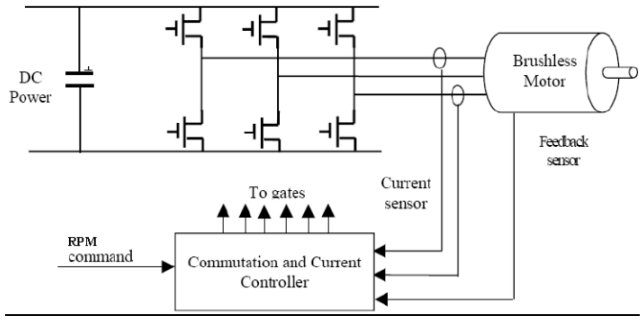


Figura 24. - Diagrama de controle de rotação em malha fechada

Fonte: Challapalli,e Gupta (2008).

Como a rotação é controlada a partir da medida da velocidade angular do rotor, será implementado um PWM por meio de um dos contadores (*timer*) internos

do controlador, que comutará os MOSFETS em frequência do PWM com ciclo útil (*duty cycle*) próxima de 4 kHz, de acordo com a necessidade de atuação exigida pelo algoritmo de controle. Consegue-se, assim, por meio de uma tensão média gerada pelo PWM, a intensidade de corrente necessária para propiciar uma velocidade de rotação estipulada pelo controle externo da roda de reação por meio da interface serial. Um fluxograma simplificado do controle efetuado pelo micro-controlador é mostrado na Fig. 2.5.

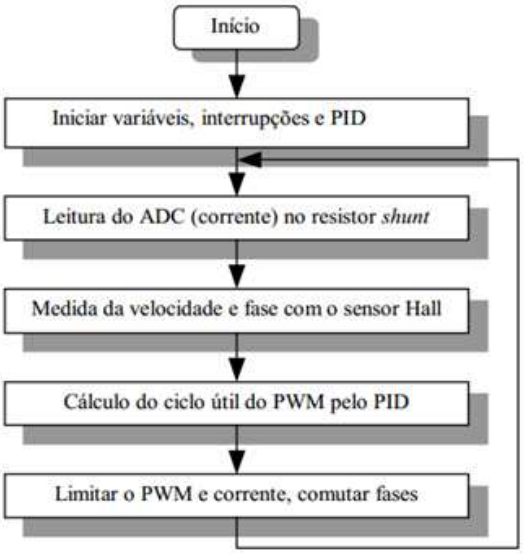


Figura 2.5.- Fluxograma do programa a ser implementado no micro-controlador.

3. PROJETO DA ELETRÔNICA DE CONTROLE

Com o emprego de diversas técnicas de controle do motor BLDC, que podem contar tanto com sensores Hall quanto sensores ópticos, ou até mesmo com métodos de controle *sensorless* (sem sensores), será possível contar com uma plataforma de desenvolvimento versátil, que permite aumentar a redundância nas medições, otimizar a utilização dos sensores, e determinar as melhores faixas de operação, por exemplo. Este projeto pode, inclusive, ajudar na seleção de sensores que facilitem o controle em baixas rotações e na inversão de sentido de rotação quando a velocidade angular for nula.

A Fig 3.1 mostra a placa eletrônica já finalizada e montada, cujo projeto será descrito nas próximas seções. Mais detalhes podem ser encontrados no Apêndice A, que apresenta o esquema elétrico do circuito eletrônico de acionamento.

Figura 3.1. - Foto da placa controladora montada. Na esquerda mostra-se o detalhe da fonte de alimentação e do conector do motor. Na direita aparece o circuito de acionamento e de processamento.

Devido às características do tipo de acionamento das fases, isto é, devido aos transistores MOSFET em *high side* e *low side* estarem alinhados em série, foram implementados circuitos analógicos extras de proteção contra inversão de fase e contra o acionamento acidental de múltiplos.

O projeto do controlador permitirá programar o acionamento do motor tanto em linguagem C usando o microcontrolador como também em linguagem VHDL utilizando um componente FPGA (do inglês *Field-Programmable Gate Array*, ou arranjo de portas programável em campo, de nome Altera Cyclone II).

Essa duplicidade de processamento permitirá tanto implementar o controle ou parte do controle na FPGA quanto no micro-controlador. Para a comunicação, controle e parametrização do controlador, uma porta serial e outra porta USB permitirão tanto o monitoramento do sistema em tempo real quando a mudança de pontos de operação (*set-points*), incluindo os ajustes necessários para um controle dedicado (*stand-alone*) embarcado.

Nesse circuito eletrônico de acionamento, no entanto, precisa-se essencialmente utilizar o micro-controlador para ler os valores de tensão sobre o resistor de shunt, sendo esta a única forma disponível para ler valores analógicos. Na Fig. 3.2 observa-se o circuito eletrônico acoplado ào volante de inércia.

Figura 3.2 – Foto do controlador eletrônico finalizado acoplado ao volante de inercia pronto para os ensaios.

3.1. FPGA

Para o FPGA foi escolhido o circuito integrado Altera Cyclone II EP2C8 de 144pinos com 8,256 elementos lógicos. Essa escolha foi feita pela facilidade de se encontrar e conseguir a documentação necessária para o projeto, operação e programação do componente, embora a pesquisa de componentes também tenha levado em conta a disponibilidade de componentes Hard-Rad (com proteções a efeitos de radiação). A grande dificuldade de se conseguir FPGA Hard-Rad e seu custo elevado inviabilizou sua escolha nesse trabalho.

A incorporação do FPGA no projeto visa oferecer maiores recursos de programação e controle, uma vez que filtros e condicionamento de sinais digitais exigem uma alta velocidade de processamento e consomem muito tempo dos micro-processadores. O interessante nesse caso é que se pode implementar, testar e validar as rotinas no micro-controlador e no FPGA e, após terem sido validados, pode-se gerar um circuito integrado customizado para a aplicação (ASIC, do inglês *Application Specific Integrated Circuit*, ou circuito integrado para aplicação específica), embora a aplicação e produção de um ASIC só se justifique para grandes produções.

O FPGA faz uma interface direta dos sinais de acionamento do micro-controlador com o motor. Essa arquitetura permite uma infinidade de aplicações, e dentre elas citam-se: criação de filtros, maior poder de cálculo, condicionadores de sinais, etc. Além disso, ele pode igualmente ser programado para fazer interface nos sinais de sensores possibilitando novos tratamentos digitais de sinais.

Na implementação realizada neste trabalho o FPGA será programado somente como um *by-pass* dos sinais de acionamento e condicionamento das fases, pois todo controle será feito pelo micro-controlador nesse momento.

A Fig. 3.3 ilustra o diagrama elétrico parcial do FPGA projetado, no qual são definidos os terminais de conexão entre o microcontrolador e as entradas e saídas dos sinais.

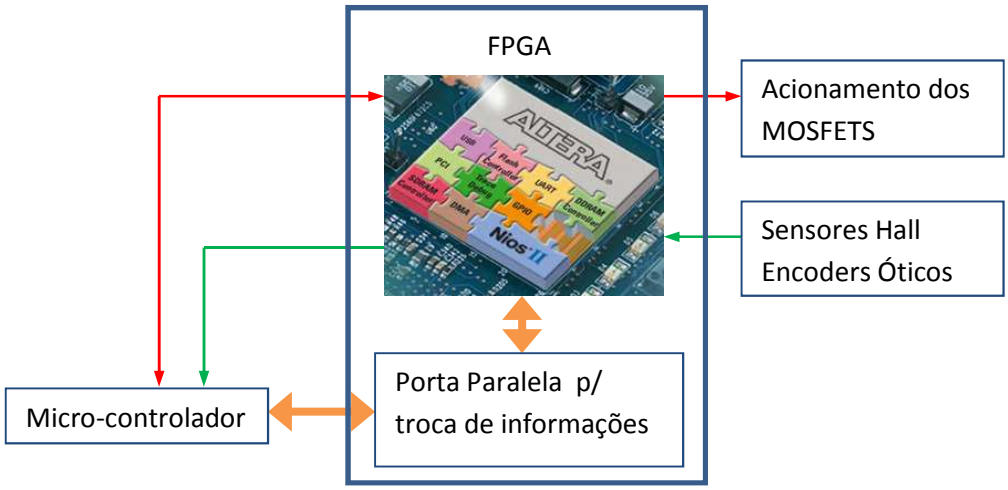


Figura 3.3 - Diagrama em bloco das conexões de entrada e saída do FPGA.

3.2. Microcontroladores

Além de possuir um FPGA, a placa conta também com um micro-controlador de 32 bits programado em linguagem C que pode tanto ser um ARM SAM4S quanto um PIC32MX, bastando apenas soldar qual dispositivo pretende usar em sua posição demarcada na placa já que se tem duas posições diferentes para soldagem na placa relativas a cada um dos processadores. Apenas um processador é aceitável em conjunto com a FPGA.

Tanto o diagrama esquemático quanto a placa de circuito impresso foram concebidos para que se possa escolher qual dos dois micro-controladores se deseja instalar e usar (um ou o outro). Isso garante flexibilidade no projeto, e permite estudos comparativos de procedimentos em processadores diferentes, além de viabilizar processamento simultâneo com o FPGA e um dos micro-controladores, com alto poder de processamento e grande suporte de ferramentas de desenvolvimento já existentes.

Usando técnicas seguras de programação, na qual o *software* (programa de aplicação) é separado do *firmware* (programas de interface da aplicação com o *hardware*) pode-se ter um único *software* para os 2 diferentes tipos de micro-controladores, como será explicado no capítulo 4.

Um dos micro-controladores usado é o de núcleo M4 de 32bits com frequência de 120 MHz fabricado pela Atmel. Seu código é ATMEL ARM SAM4S. Esse micro-controlador é capaz de adquirir todas as informações analógicas dos sensores de corrente do motor por meio de 16 entradas analógicas de 12 bits e também possui entradas para sinais digitais dos sensores de rotação, tanto o codificador óptico (*encoder*) quanto o sensor de efeito Hall.

A escolha desse processador foi baseada no seu alto poder de processamento e também pelo fato de a Atmel produzir, nesta linha de processadores, micro-controladores *Rad-Hard*. Mesmo que estes não sejam compatíveis, ainda assim ele permite uma familiarização com as ferramentas de trabalho e programação. O esquema elétrico deste micro-controlador é mostrado na Fig. 3.4.

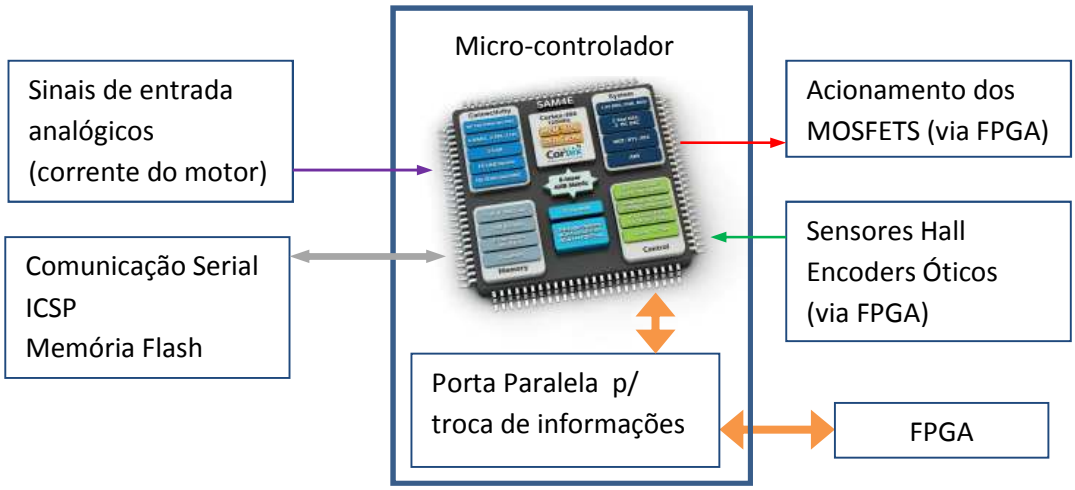


Figura 3.4 - Diagrama em bloco das conexões do micro-controlador ARM SAM4S.

O outro micro-controlador que poderá ser usado no lugar do ARM é um processador com núcleo de 32 bits e frequência de operação de 80 MHz fabricado pela Microchip, de código Pic32MX675. Esse micro-controlador também é capaz de adquirir todas as informações analógicas dos sensores de corrente por meio de 16 entradas analógicas de 12 bits com DMA (*Direct Memory Access*, ou acesso direto à memória) e, analogamente, também as entradas de sinais digitais dos sensores de rotação.

Este processador foi escolhido devido à facilidade de se encontrar e usar as ferramentas de *software* e *hardware* para programação e depuração. O esquema elétrico das conexões deste micro-controlador é ilustrado na Figura 3.5.

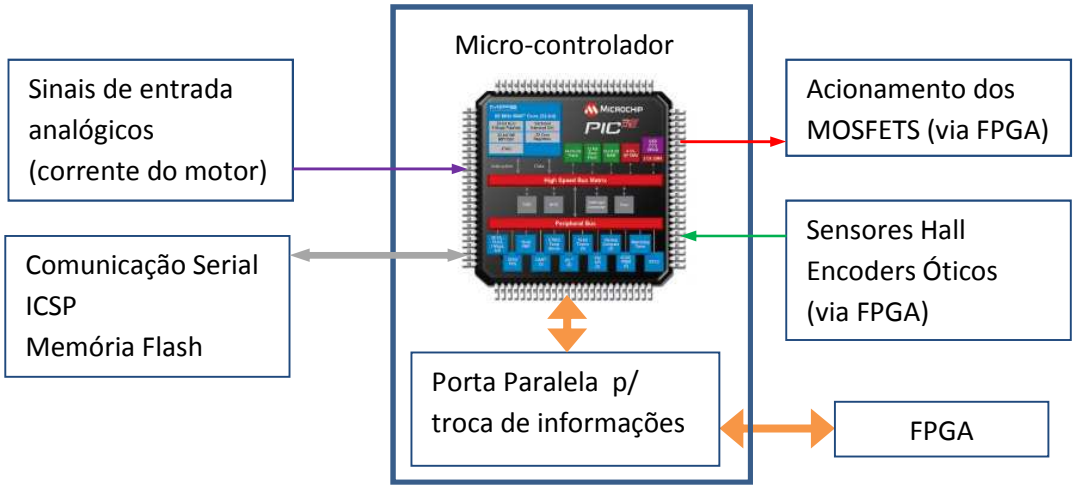


Figura 3.5 - Diagrama em bloco das conexões do micro-controlador PIC32MX.

3.3. MOSFETS

Como não se deseja limitar os testes e ensaios futuros, decidiu-se acrescentar um número maior do que o necessário de circuitos de acionamento de fases do motor *brushless*, ou seja, de pernas da ponte H. A placa conta então com 5 saídas de alimentação independentes para 5 possíveis ligações de fase. Isso quer dizer que é possível utilizar motores com até 5 fases ligados a esta placa, mas também é igualmente possível ter-se motores com 2, 3, ou 4 fases. Um número maior de possibilidades no acionamento de fases também permite uma redundância e um aumento na confiabilidade do circuito, já que qualquer circuito de acionamento pode ser substituído por outro facilmente, no caso de se contar com um motor com menos de 5 fases.

A Fig. 3.6 apresenta o esquema elétrico do circuito de acionamento das fases, com 5 pernas de ponte H.

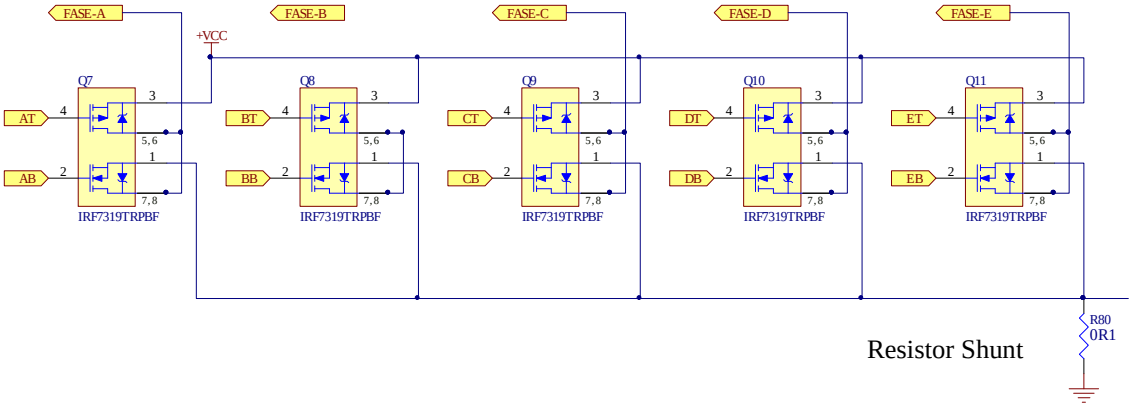


Figura 3.6 - Detalhe do esquema elétrico dos 5 pares de transistores MOSFETS.

Foram inseridas no projeto eletrônico medidas de segurança analógica por *hardware* de modo a se evitar altas correntes elétricas entre os circuitos de MOSFETS do *high-side* e *low-side.* Estas medidas são descritas a seguir.

3.4. Segurança dos MOSFETS

Para evitar que haja um acionamento simultâneo nas portas de ambos os transistores MOSFETS de uma dada perna da ponte H, isto é, de um par *high-side* e *low-side,* introduziu-se um circuito formado por duas portas lógicas *And* (E) e um inversor nas entradas dos transistores, como mostra o diagrama da Fig. 3.7, de tal forma que, com apenas um sinal de entrada pode-se escolher seletivamente qual transistor será acionado, garantindo, com isso, que o outro permaneça inativo. As portas *and* também permitem a incorporação de um sinal de habilitação (*enable*) que ativa ou desativa o par de transistores.

Essa proteção é de grande importância quando se tem uma placa que será usada em ensaios e testes, na qual o *software* que controla os transistores MOSFET é executado fora das bases de tempo do micro-controlador, em um processo de *debug*, por exemplo.

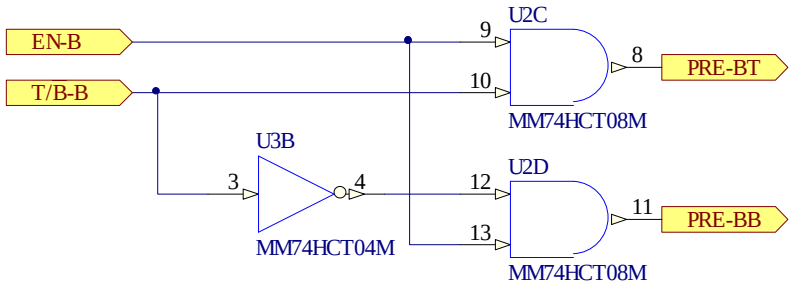


Figura 3.7 -Detalhe do circuito elétrico de proteção dos MOSFETS.

3.5. Acionamento dos MOSFETS

O acionamento dos MOSFETS é feito em conjunto com a proteção mencionada anteriormente e foi elaborado utilizando transistores bipolares de uso comum para a parte *high side* onde se deseja garantir um *pull-up* sem ter a referência no sinal de aterramento (GND) e alguns componentes passivos como resistores limitadores de corrente e *pull-down* no caso das portas (*gates*) dos mosfets *low side*, mostrado esquematicamente na Fig. 3.8.

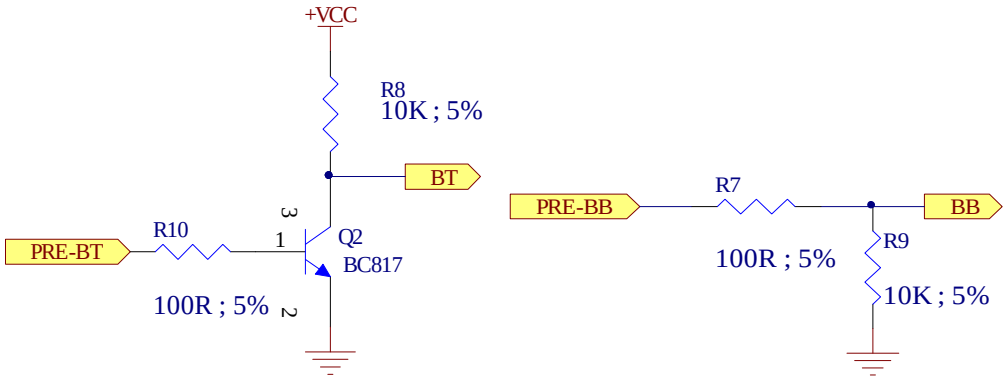


Figura 3.8 - Esquema do circuito de acionamento dos transistores MOSFET.

Para reduzir as emissões eletromagnéticas (EMC, do inglês *Electromagnetic Compatibility*, ou compatibilidade eletromagnética) e as oscilações harmônicas de alta frequência, foram introduzidos no circuito filtros LC (indutor-capacitor) entre as portas de saída dos MOSFETS e as fases do motor *brushless* (Rajan, 2009), conforme é apresentado no diagrama da Fig. 3.9.

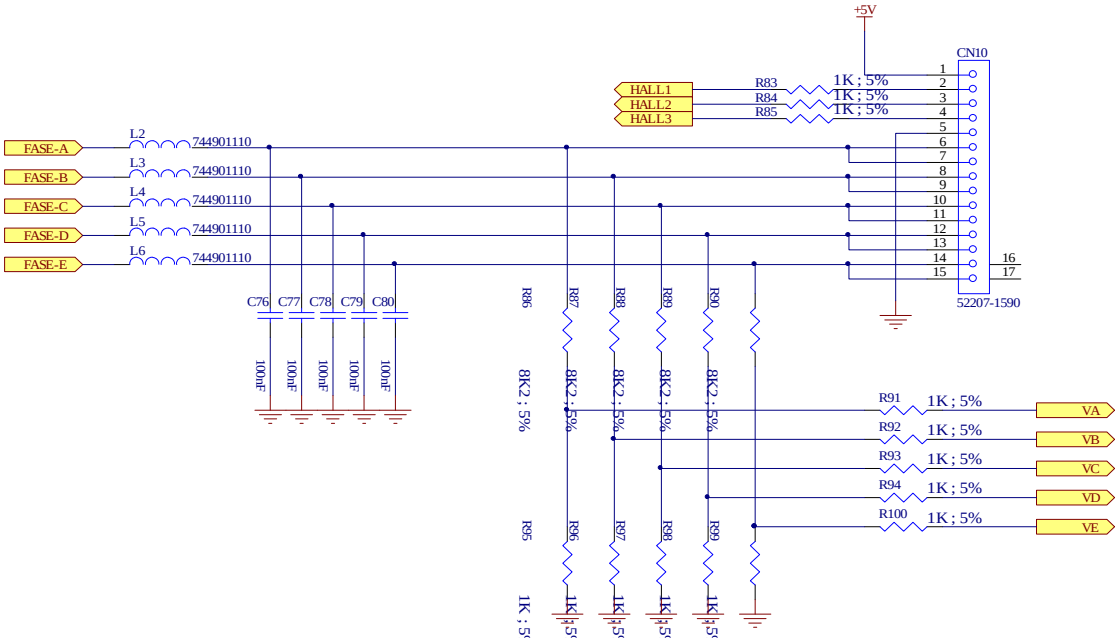


Figura 3.9 - Detalhe da conexão do motor, filtro LC e leitura das tensões das fases.

3.6. Sensores

O circuito de leitura de corrente é composto por um resistor de shunt de precisão com baixíssimo valor (0,1 Ohm), no qual se efetua a medida de tensão devido à passagem da corrente total das fases do motor. Um amplificador operacional foi acrescentado ao circuito para amplificar a tensão de forma a viabilizar a leitura pelo conversor ADC (*Analog to Digital Converter*, ou conversor análogo-digital) do micro-controlador.

O amplificador foi configurado como não inversor, com ganho de 50, aproximadamente, resultando em uma tensão de leitura de 3 V máximos na saída do amplificador, conforme ilustra a Fig. 3.10.

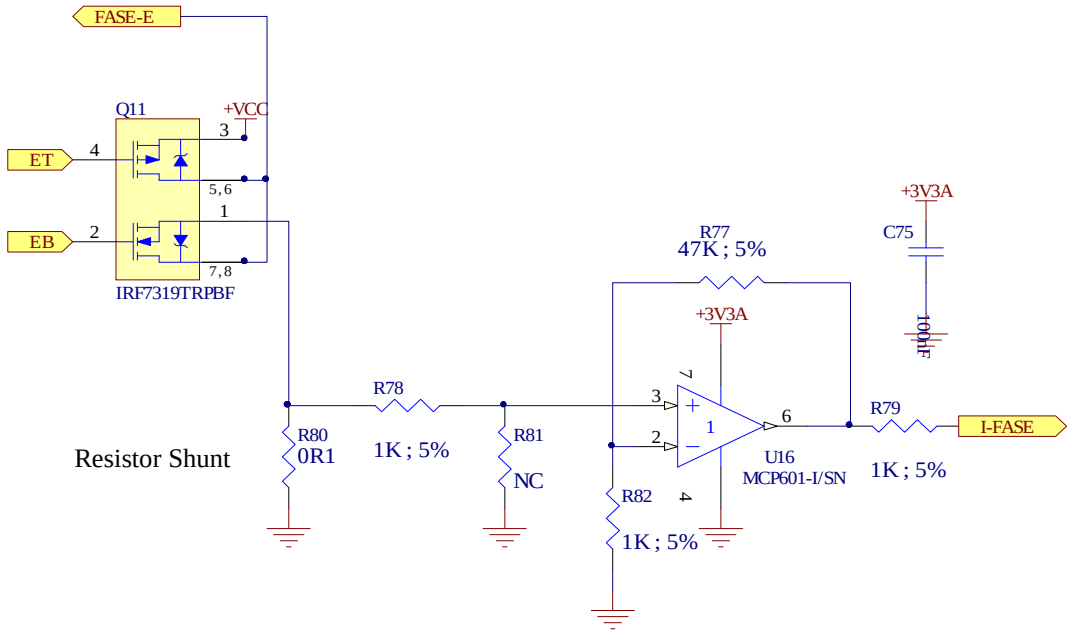


Figura 3.10 - Amplificador operacional de leitura da corrente.

Essa capacidade de leitura da corrente é de extrema importância quando se pretende fazer um controlador sem sensores ópticos ou de efeito Hall, denominados de *sensorless.*

O processador tem acesso às entradas analógicas de 12 bits com um acesso direto à memória sem a necessidade de interromper o programa principal para chamar rotinas de aquisição de dados para obtenção das leituras. Essa função garante um bom funcionamento do controle em malha fechada, já que os dados são adquiridos em taxa de amostragem fixa e prioritária.

Para se conseguir um controle em malha fechada é necessário conhecer a posição do rotor para alimentar as fases do motor corretamente. Para isso far-se-á uso dos sensores Hall (no caso do motor utilizado neste trabalho, 3 sensores) instalados internamente no motor que foi cedido pelo Instituto Mauá de Tecnologia. A Fig. 3.11 apresenta as principais medidas e características do motor utilizado.

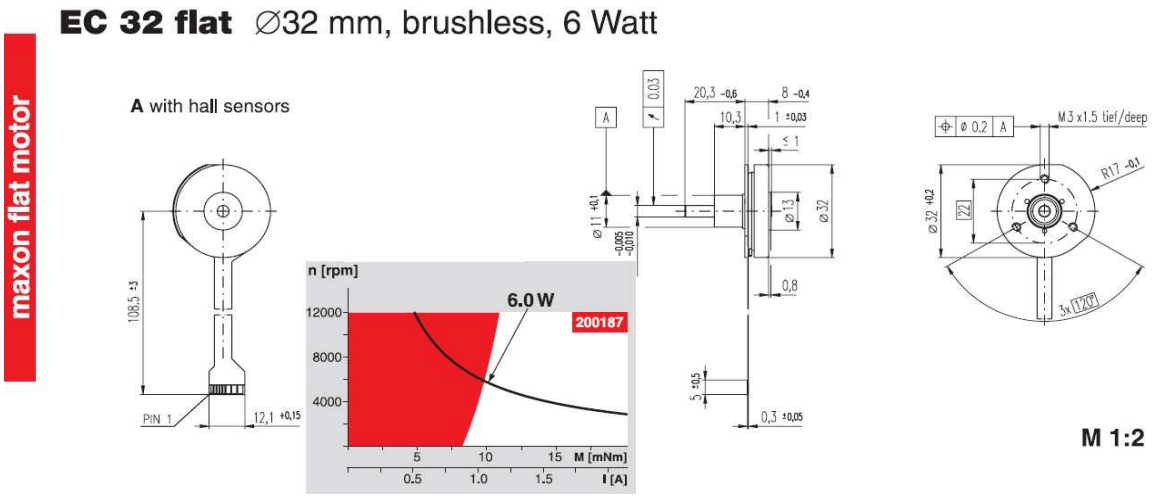


Figura 3.11 - Motor brushless com a posição dos sensores Hall.
Fonte: Maxon Motor (2014).

Além do sincronismo obtido pelos sensores Hall, o projeto possibilitará o uso de um disco graduado que, em conjunto com sensores ópticos, poderão melhorar a resolução na posição do eixo rotorA Fig. 3.12 apresenta o circuito de condicionamento do sinal para os sensores.

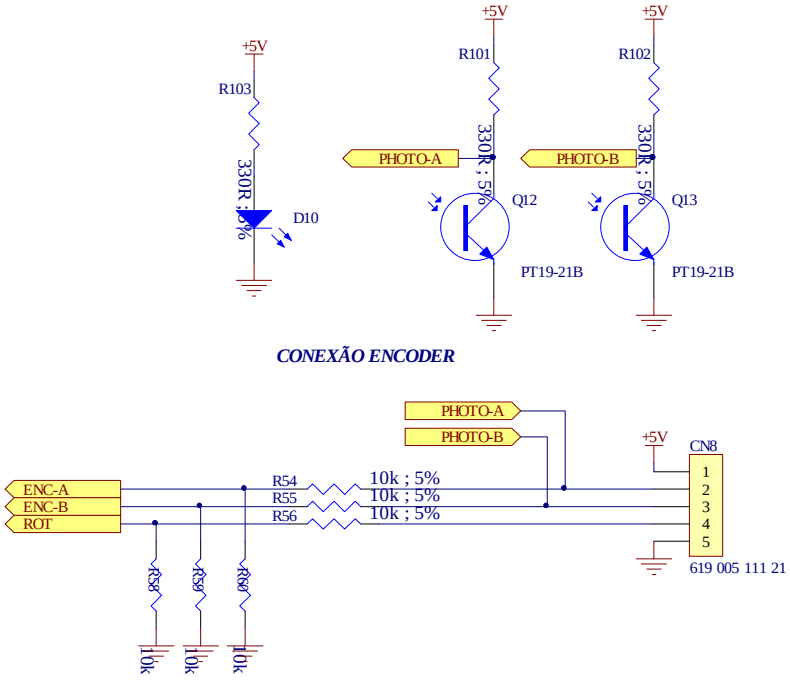


Figura 3.12 - Circuitos de entrada dos sensores Hall e *encoders*.

Para a indicação de sentido de rotação, além do sensor de efeito Hall, também será disponibilizada uma entrada para *encoders* digitais que podem ou não ser incorporados ao disco graduado. Por fim, uma entrada extra para mais um sensor hall estará disponível, que poderá ser usada caso o rotor de inércia utilize um disco metálico dentado para efetuar a medida de velocidade angular.

3.7. Comunicação

Para se comunicar com o exterior a placa foi dotada de conexão ICSP para gravar o programa em C, e uma conexão JTAG utilizada para depuração do programa e para gravar programas no FPGA. O projeto da placa incorpora ainda saídas seriais para habilitar comandos e para permitir o monitoramento remoto por uma outra unidade computacional, como um PC, por exemplo. Essa conexão remota permitirá ajustar os pontos de operação (*set-points*) de operação da roda de reação. Foi criada também uma porta paralela entre o FPGA e o micro-controlador, para que se possa transmitir dados entre as unidades de processamento. O circuito de comunicação é apresentado na Fig. 3.13.

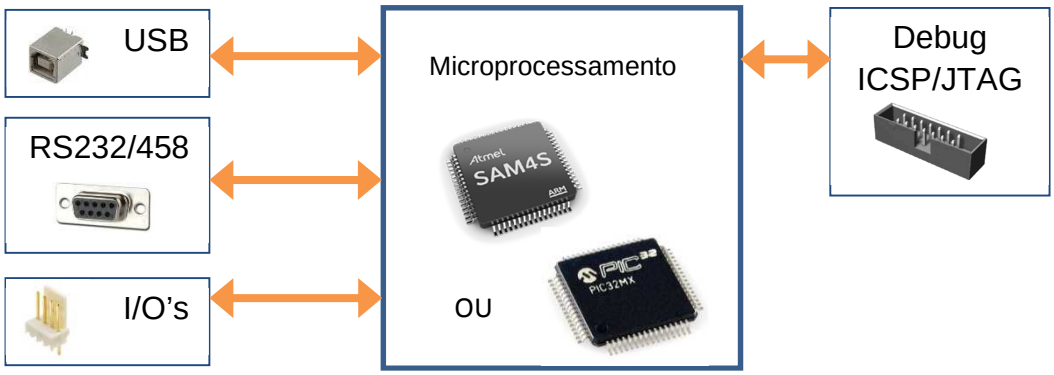


Figura 3.13 - Diagram em blocos da interface de comunição.

Como o acesso ao programa básico será permitido, então a inclusão de novos controladores, a gravação de novos ajustes de pontos de operação em memória não volátil e até mesmo a alteração do modo de comando atual será facilitada por meio das interfaces de comunicação.

3.8. Fonte de alimentação

Na parte de alimentação foi utilizado um circuito de fonte chaveada, e de larga capacidade de lidar com diferentes tensões de entrada (de 7 a 30 Volts) e geração de corrente (até 1A DC). Possibilita com isso a versatilidade do projeto para um amplo espectro de motores existentes. A Fig. 3.14 ilustra o circuito de alimentação.

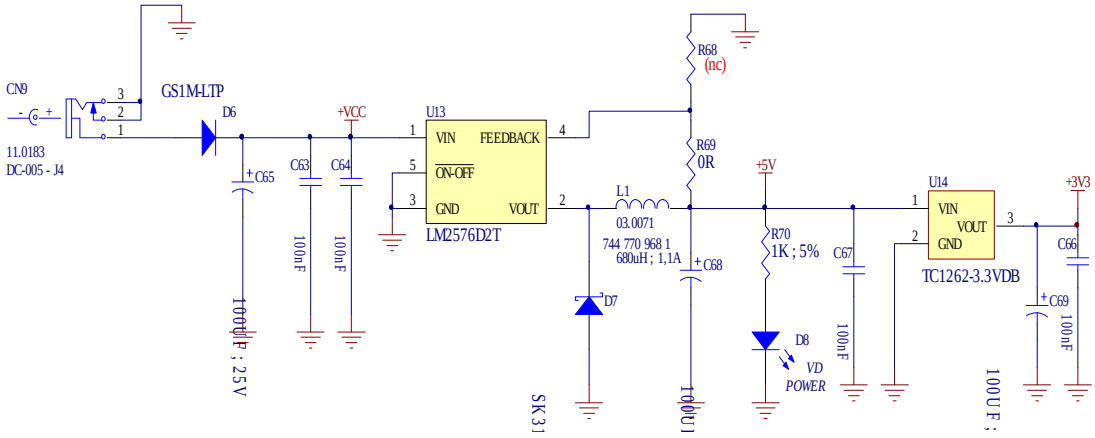


Figura 3.14 - Esquema elétrico da fonte de alimentação.

3.9. Placa de circuito impresso

Para a confecção da placa de circuito impresso foi utilizado o software Altium Designer e o resultado pode ser visto na Fig. 3.5 O arquivo contendo o esquema em “gerber” da placa é apresentado no Apêndice B - Visão da placa controladora.

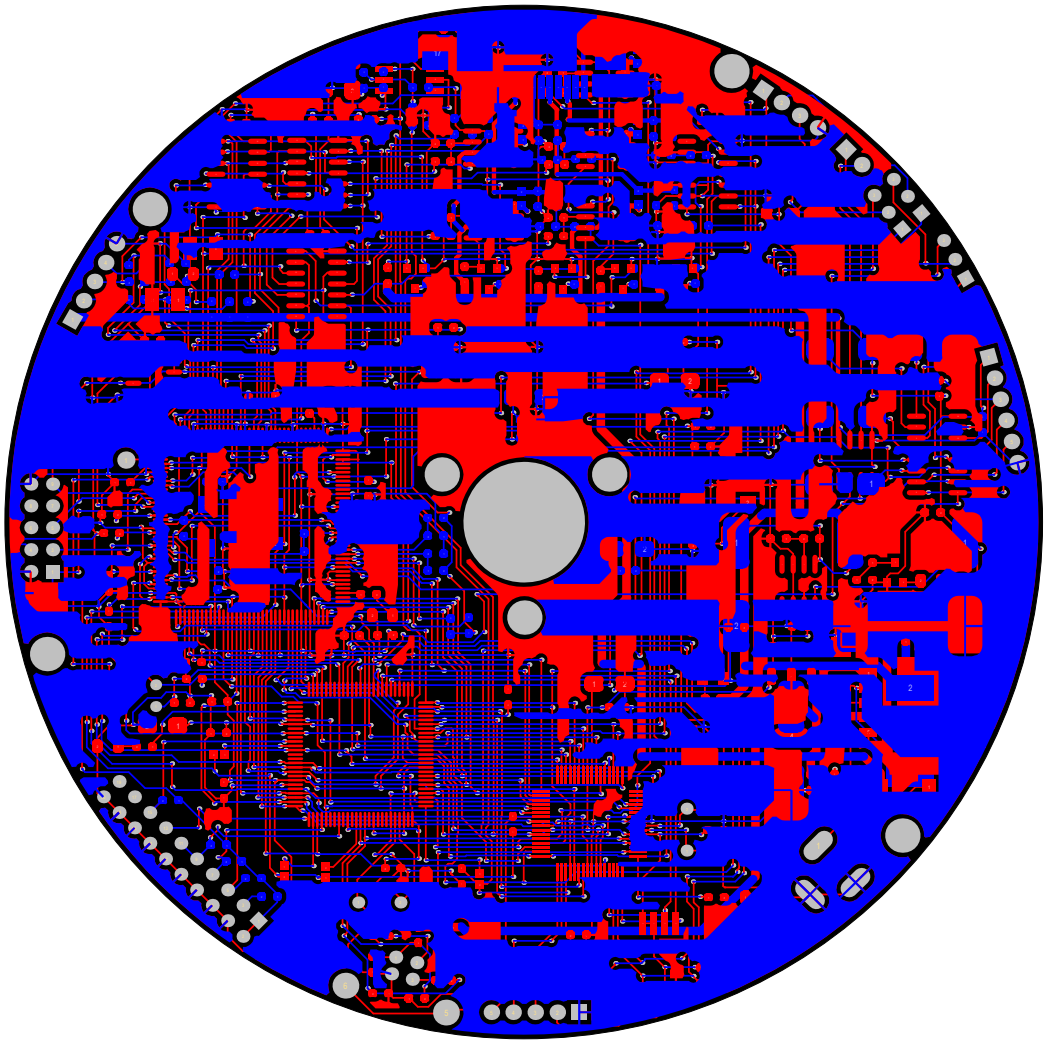


Figura 3.15 - Placa de circuito impresso do projeto da roda de reação.

4. PROGRAMA DE CONTROLE DA RODA DE REAÇÃO

O programa executado no microcontrolar foi escrito em linguagem C e será descrito a seguir. Para ter o correto acionamento das fases do motor sem escovas deve-se ajustar o sincronismo do acionamento das fases com a posição do eixo rotor do motor. Esta necessidade deve-se ao modo de construção do motor, que possui 3 sensores Hall distanciados de 120° entre si, que fornecem a referência da fase a ser acionada a cada instante. Além da referência para acionamento das fases, os sensores Hall também fornecem meios para se medir a velocidade de rotação do eixo do motor.

Os sensores Hall são detectados por meio de interrupções do microcontrolador. O sensor muda de estado (de 0 para 1) quando um campo magnético intenso aproxima-se ou afasta-se do sensor. As interrupções detectam portanto a passagem do campo magnético induzido por um imã fixado ao rotor, e realizam um incremento na máquina de estados que realiza o acionamento das fases do motor. Nesta máquina de estados é armazenada a tabela de acionamento do MOSFETS.

Para tal propósito a máquina de estados executa o chaveamento com base numa tabela de acionamento para os transistores MOSFET *high side* e outra tabela para os transistores inferiores:

```
tab_mosfet_sup[]={0b100, 0b010, 0b010, 0b001, 0b001, 0b100}
tab_mosfet_inf[]={0b001, 0b001, 0b100, 0b100, 0b010, 0b010}
```

Ao acionar o mosfet superior do 3º bit o mosfet inferior do primeiro bit (usando índice 0 para as tabelas acima) será fechado o circuito para a passagem de corrente elétrica por uma fase do motor. Essa tabela pode ser modificada para que o motor seja acionado de forma conveniente no caso de ensaiar um outro motor que necessside de outra sequência de acionamento de fases.

O circuito de proteção só entrará em ação caso um mosfet superior e inferior do mesmo bit sejam acionados simultaneamente. Observa-se que dos 5 bits

necessários para o acionamento do circuito, apenas os 3 menos significativos foram usados, devido ao fato do motor ter apenas 3 fases. Além disso, o bit menos significativo representa o par de MOSFET 1 e o mais significativo representa o MOSFET 5.

Para o correto funcionamento do circuito não se deve ter um mesmo bit das tabelas superior e inferior ligados na mesma posição indexada nas tabelas. Outro detalhe que deve ser respeitado é a introdução de um atraso entre as trocas de acionamentos, para acomodar os tempos de resposta dos transistores MOSFETS (RASHID, 2001). Para os MOSFETS utilizados (IRF7341) esse tempo foi definido em 100ns, isto é, após uma fase ser desconectada, será garantido um tempo de 100ms até um novo acionamento de fase.

Após o acionamento dos transistores de acordo com a posição do rotor do motor, deve-se agora modular a tensão aplicada em cada fase de modo a ajustar a corrente fornecida ao motor. Isto é conseguido utilizando um PWM no acionamento da fase.

A geração do PWM é realizada também por interrupção, porém agora este é comandado por um contador temporizado (*timer*) do micro-processador. A função de interrupção gera na porta dos transistores MOSFET um sinal com frequência de 4kHz cujo ciclo útil (*duty cycle*) varia conforme o valor da intensidade desejada na corrente do motor. O *duty-cycle* descreve a fração de tempo em que o MOSFET estará no estado "ativo", ou seja, estará ativando uma das fases do motor. O corte da corrente é efetuado pela interrupção dos sinais apenas nos transistores MOSFET superiores. Isso permite que os MOSFETS inferiores, efetuem a recirculação da corrente através de seu diodos internos reversos, que é criada pelo campo magnético ainda existente nos enrolamentos do motor no momento do corte da corrente elétrica nos mosfets, causada pelo PWM.

O ajuste do *duty-cycle* do PWM é feito pelo microprocesador, de acordo com o algoritmo de controle PID, cuja referência é a velocidade de rotação que se

deseja ter no eixo do motor e a variável de controle é a velocidade medida no eixo pelos sensores (Hall ou *encoders*). A saída do controle PID é exatamente o sinal que será utilizado para atualizar o PWM de acionamento dos mosfet, que varia de 0 a 100% de ciclo útil. Ou seja, de totalmente desligado a totalmente ligado.

O controle do motor pode igualmente ser realizado por meio da corrente, no qual é estabelecido uma corrente de referência que se deseja, e o sinal de realimentação é obtido pela leitura da tensão no resistor de *shunt* após ter sido amplificado pelo circuito do amplificador operacional. A corrente de controle na saída do algoritmo de controle PID também é aplicada na atualização do PWM. Ressalta-se que apenas um controle é ativo a cada instante e sua seleção é feita via protocolo serial através de comando originado no PC conectado à roda de reação.

A leitura dos sinais analógicos é feita por funções de interrupção do timer com período de 1ms e, após uma média simples de 64 amostras, as medidas são armazenadas na estrutura de registro do programa.

A implementação do controle PID segue a equação (ÅSTRÖM,1995):

$$u(t) = Kp.e(t) + Ki\int_0^t e(t)dt + Kd\frac{d}{dt}e(t) \qquad (4.1)$$

onde K_p, K_i e K_d são os ganhos proporcional, integral e derivativo, respectivamente, porém o código do controle PID implementado em C foi baseado no código computacional discretizado apresentado a seguir, retirado do capítulo 3.7 de (Astrom,1995).

```
"Compute controller coefficients
          bi=Ki*h/TT              "integral gain
ad=(2*Td-N*h)/(2*Td+N*h)
bd=2*Kd*N*Td/(2*Td+N*h)       "derivative gain
aO=h/Tt
"Control algorithm
r=adin(chl)                   "read setpoint from chl
y=adin(ch2)                   "read process variable from ch2
```

```
P=Kp*(b*ysp-y)                "compute proportional part
D=ad*D-bd*(y-yold)            "update derivative part
v=P+I+D                       "compute temporary output
u=sat(v,ulow,uhigh)           "simulate actuator saturation
daout(chl)              "set analog output chl
I=I+bl*(ysp-y)+a0*(u-v)       "update integral
yold=y                        "update old process output
```

Após a sintonia empírica do sistema baseada no método Ziegler-Nichols (ÅSTRÖM,1995), os coeficientes são:

```
#define PID_SPEED_N           20.0
#define PID_SPEED_H           40.0
#define PID_SPEED_TT          40.0
#define PID_SPEED_KP          500.0
#define PID_SPEED_KI          10000.0
#define PID_SPEED_KD          0.0
```

O código completo em C da placa de controle pode ser visto no Apêndice E.

Para que o sistema de controle seja utilizável tanto em modo de controle de corrente quanto em modo de controle de velocidade angular será efetuada uma troca na instância do controlador utilizado, isto é, ter-se-ão dois conjuntos de posições de memória onde serão armazenadas as variáveis de cálculo, tornando possível a troca imediata do modo de controle sem que se necessite de uma nova inicialização das variáveis acumuladoras das iterações anteriores. No entanto, essa troca de modos de controle se mostrou interessante para chaveamentos em condições constantes de rotação ou corrente, onde o controle está estabilizado.

Para a operação da placa de controle foi elaborado um protocolo de comunicação por meio da interface serial pela qual se podem ajustar os pontos de referência para a velocidade angular e a corrente, bem como os parâmetros de ajuste do PID. Além disso, o protocolo também retorna mensagens de aceite de comando e envia telemetrias com os valores correntes da velocidade de rotação e da corrente, de forma a permitir o fechamento de uma malha para o controle de atitude.

O protocolo das mensagens implementadas segue o padrão SunSpace referente ao documento Reaction Wheels SSIS Users Manual:

```
//Header    Address     Packet ID   Data
//0x7F      0xF1       0x40         0x00    0x00    0x7A    0x44
```

onde `Header`, `Address` e `Packet_ID` representam o cabeçalho, o endereço do dispositivo em questão, o tipo de mensagem, e o campo `Data` informa o valor a ser enviado ou recebido. Para uma simples resposta de aceite de um comando o dispositivo apenas responde a mesma mensagem, porém sem transmitir o campo Data.

Para a comunicação do micro-controlador com o FPGA foi projetada uma porta paralela tanto no micro-controlador quanto no FPGA. Assim, foi possível criar um mapa de registro de entrada e saída onde tudo o que acontece na placa passa a ser compartilhado entre o FPGA e micro-controlador. Embora o FPGA só tenha sido programado para fazer um “by-pass” nos pinos de entrada e saída entre micro-controlador e sensores e atuadores, a porta paralela e o mapa de registros serão de grande valia na troca de informações.

Um ponto de grande importância no funcionamento do controle do motor é reconhecer o instante em que o rotor do motor passa pelo sensor Hall, ou seja, o momento em que este envia um sinal digital para o micro-controlador. Para a eficiência do algoritmo é necessário que o micro-controlador seja interrompido no instante em que recebe o sinal do sensor Hall. Portanto, as saídas dos sensores Hall devem ser conectadas aos pinos de interrupção do micro-controlador. Como há 3 sensores Hall, tem-se também 3 interrupções distintas, porém, como o sistema de chaveamento das fases baseia-se em 6 passos, faz-se com que cada interrupção seja ativa tanto na borda de subida quando na borda de descida do sensor, causando assim 2 interrupções para cada um. Isso é possível alterando a sensibilidade da interrupção para interromper o programa a cada troca de nível de sinal.

A seguir apresenta-se o código parcial das interrupções que registram o momento da mudança de estado de cada sensor e informam ao programa de controle das fases que o índice na tabela de acionamento dos MOSFETS deve ser trocado.

```
void __ISR(_EXTERNAL_1_VECTOR, IPL7) INT1Interrupt( void )
{
    if (INTCONbits.INT1EP == 1)
    {        executa_sensoreamento_hall( 1 );    }
    else
    {        executa_sensoreamento_hall( 0 );    }
    INTCONbits.INT1EP   =   ~INTCONbits.INT1EP;//   inverte   borda   de
interrrupção
    IFS0bits.INT1IF = 0;                        //    Limpa    flag    de
interrupção Input Capture 1
}

void __ISR(_EXTERNAL_2_VECTOR, IPL7) INT2Interrupt( void )
{
    executa_sensoreamento_hall( 0 );
    INTCONbits.INT2EP   =   ~INTCONbits.INT2EP;//   inverte   borda   de
interrrupção
    IFS0bits.INT2IF  =  0;                              //  Limpa  flag  de
interrupção Input Capture 2
}

void __ISR(_EXTERNAL_3_VECTOR, IPL7) INT3Interrupt( void )
{
    executa_sensoreamento_hall( 0 );
    INTCONbits.INT3EP   =  ~INTCONbits.INT3EP;  //  inverte  borda  de
interrrupção
    IFS0bits.INT3IF = 0;                    // Limpa flag de interrupção
Input Capture 3
 }

 inline void executa_sensoreamento_hall( unsigned char resincroniza )
{
    TMR4  = 0x0000;                  // Zera contador do timer de erro
    atualiza_rpm ( resincroniza ); //   realiza   o   chaveamento   dos
mosfets
    dados.contador_hall++;          // incrementa contador para cálculo
de rotação
}
```

Esta última rotina, `executa_sensoreamento_hall`, realiza o reset do timer4 responsável por monitorar o giro do motor. Caso haja o estouro do timer4, uma interrupção do programa principal ocorre, informando ao software que o motor foi bloqueado ou não está funcionando corretamente devido a alguma falha. Além dessa função, a rotina ainda executa a tarefa principal que é atualizar o chaveamento das fases dos MOSFETS.

O parâmetro resincroniza na função atualiza_rpm (resincroniza) garante a fase correta alinhada com a interrupção de borda de subida do sensor Hall seja acionada. O incremento na variável `dados.contador_hall` fará com que seja

computado mais um acionamento do sensor Hall no cálculo da velocidade de rotação. O cálculo da velocidade é executado a cada 10 ms, o que resulta em uma baixa precisão em rotações baixas.

Como o motor sem escovas possui 4 pólos e 3 fases, tem-se um pulso do sensor Hall a cada 15 graus de giro no eixo do motor, e considerando apenas uma interrupção em 10 ms tem-se uma medida mínima da velocidade de rotação de 14,4 rpm. Por isso foram inseridas entradas extras na placa para permitir a inclusão de sensores do tipo encoders que poderão ser implementados futuramente para resolver o problema de baixa resolução na medição de baixas velocidades. Esse problema foi parcialmente reduzido efetuando-se uma média da velocidade de rotação com um intervalo maior de tempo nas baixas velocidades.

Na foto mostrada na Fig. 4.1 observa-se o local onde é instalado um disco graduado que poderá ser usado para melhorar a leitura da velocidade angular em baixas velocidades. Essa implementação não foi feita neste trabalho, pois este escopo focou-se nas funcionalidades do acionamento eletrônico do motor BLDC embora tal recurso seja de grande importância para trabalhos realizados em baixa rotação.

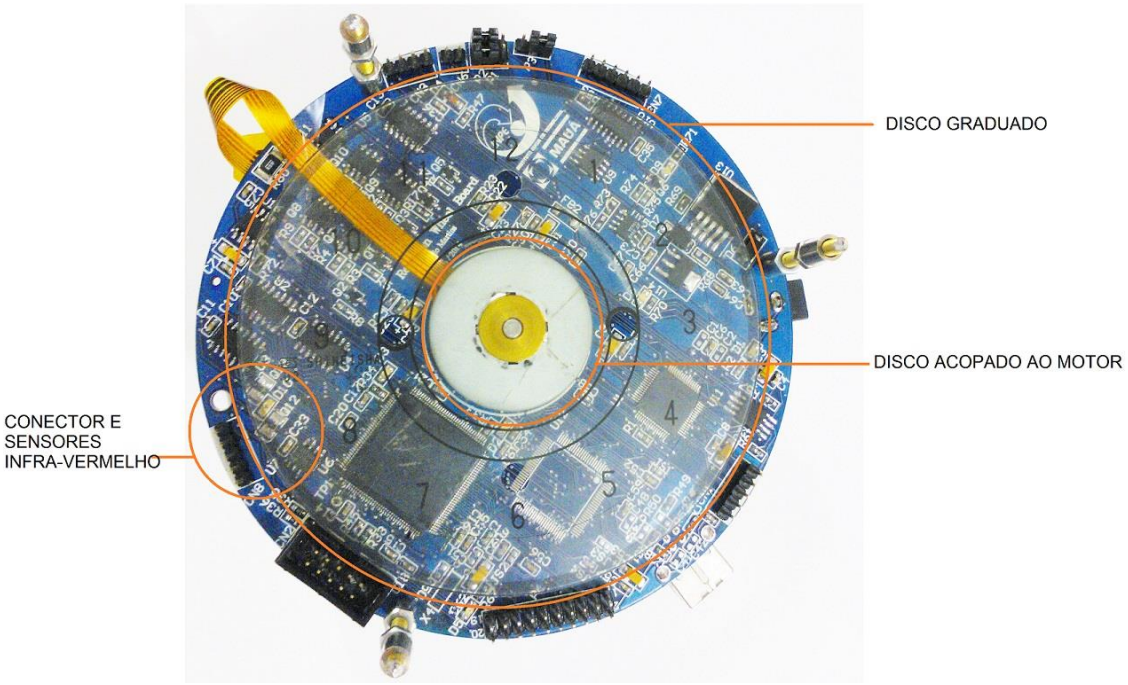


Figura 4.1 – Conectores na placa para suporte de encoder de alta resolução.

5. Ensaios realizados com o protótipo da roda

Iníciaram-se os ensaios aplicando um filtro de Kalman para a estimação dos parâmetros de atrito do conjunto rotor e massa de inércia. Neste estudo inicial é apresentado um método para estimação de parâmetros de motores CC sem escovas (MARTINS, 2012). O atrito nos mancais de rolamento torna a resposta da roda não linear, principalmente em baixas velocidades ou durante a inversão do sentido de giro. De fato, este problema é crítico, pois aumenta consideravelmente o erro de apontamento do satélite quando ocorre a inversão do sentido de rotação da roda de reação (CARRARA et al., 2011). A operação da roda de reação em modo de controle de velocidade reduz significativamente este problema, mas o projeto do controlador deve levar em conta o atrito para que este possa ser ajustado convenientemente. O conhecimento dos parâmetros de motores, portanto, é fundamental para o dimensionamento e projeto de uma roda de reação e futuramente para o controle de atitude do satélite.

Tendo essas características determinadas, foram definidos alguns ensaios para que se pudesse mensurar o funcionamento da eletrônica de controle do motor acoplado ao disco de inércia, como de fato ocorre em uma aplicação real em roda de reação. Dentre estes ensaios podem-se citar: a resposta ao degrau em velocidade, a resposta ao degrau em corrente, o tempo de decaimento da rotação, a estabilidade em rotação, a estabilidade da corrente e o diagrama de defasagem da rotação em função da frequência de comando em velocidade de rotação.

A foto da montagem da roda de reação é apresentada na Fig. 5.1, e a tela do software de validação desenvolvido em Labview é mostrada na Fig 5.2. O software auxiliou tanto para o comando dos *set-points* de corrente e RPM da roda de reação, quanto para o monitoramento e registro das variáveis em arquivo de texto no computador PC.

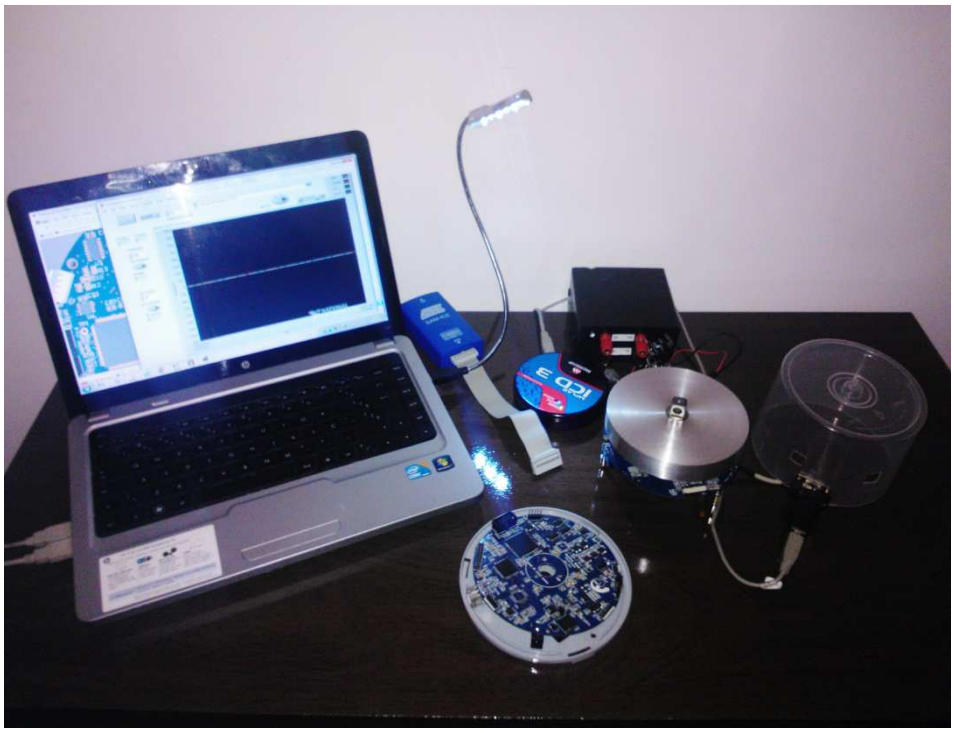


Figura 5.1 - Ambiente de validação. Software em Labview, conexões seriais e USB e fonte de alimentação.

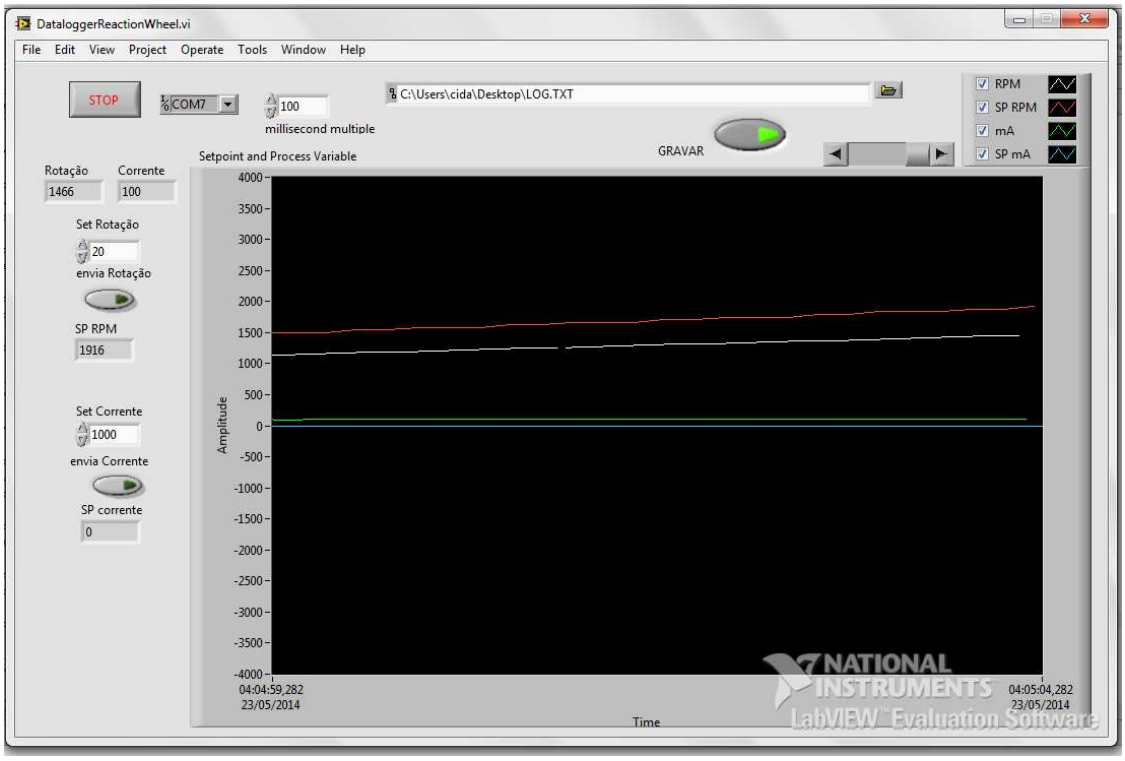


Figura 5.2- Tela do software de validação desenvolvido em Labview

Essa interface possibilita que seja alterado o alvo para o controlador, seja ele alvo de corrente ou alvo de rotação, isto é, consegue comandar a roda de reação para que ela atue na rotação ou corrente que se deseja. Além de comandar a roda de reação, o software em questão também armazena os valores de rotação e corrente em um arquivo de texto para posterior análise.

Na Fig. 5.3 é detalhada a construção e montagem do circuito eletrônico com o volante de inércia acoplado ao motor Maxxon.

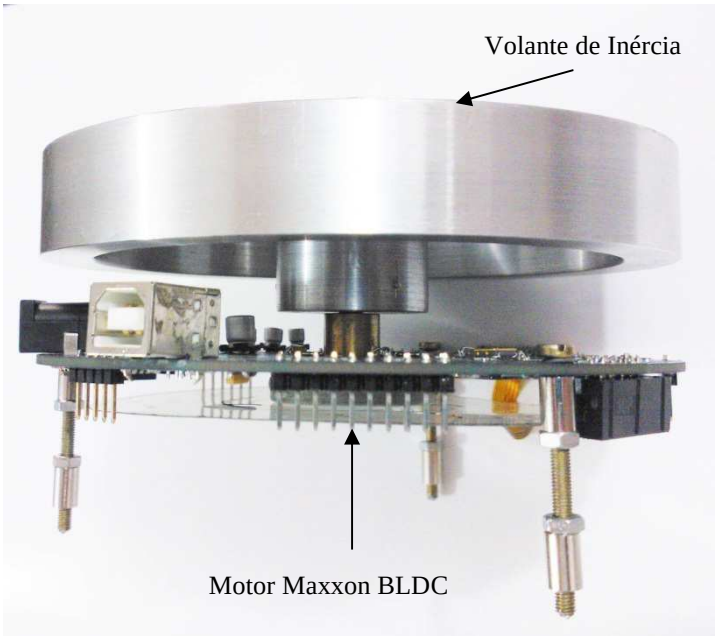


Figura 5.3- Montagem da roda de reação com o volante de inércia.

A aquisição dos dados foi feita conectando-se a placa micro-controlada a um computador padrão PC, no qual os dados são coletados e armazenados por um programa desenvolvido em Labview: tanto a velocidade de rotação do motor quanto a corrente medida naquele instante são gravadas em arquivos para posterior processamento em Matlab ou outro programa para tratamento dos dados. A taxa de amostragem é de 10Hz, ou seja, a cada 100 ms tem-se uma nova medida da velocidade angular e da corrente aplicada ao motor. Dependendo da velocidade o software de PC essa taxa pode ser aumentada já que a placa responde imediatamente ao comando de solicitação de informações de corrente e rotação.

5.1. Determinação dos parâmetros de atrito aplicando filtro de Kalman

Aplicou-se então o método de estimação de parâmetros de motores CC sem escovas para utilização no projeto da placa controladora da roda de reação. O modelo desenvolvido aplicado foi o de um motor CC sem escovas de baixo torque, com um controlador de corrente em PWM (Pulse Width Modulation). O motor foi excitado com perfis de corrente em diferentes velocidades angulares e os dados foram coletados pelo software em Labview via conexão serial com a placa controladora, sendo que a própria placa eletrônica possui um sistema de aquisição digital que mede a velocidade de rotação por meio dos sensores de efeito Hall e a corrente por meio de um resistor shunt padrão em série com as fases do motor. A corrente é determinada por meio da leitura de tensão nos terminais deste resistor. A corrente proporcional à medida de tensão sobre os terminais desse resistor. Os dados são processados por um filtro de Kalman em Matlab, que estimou a constante do motor e os torques de atrito.

Para que o filtro de Kalman possa ser aplicado e assim determinar a constante do motor e os coeficientes de atrito, deve-se coletar um conjunto de dados de modo que estes dados passem por toda a excursão de velocidades do motor necessárias para a identificação do sistema. O motor é então comandado com uma referência em velocidade de rotação com amplitude senoidal por alguns ciclos completos. Nesse ensaio coletou-se 4 ciclos completos.

Utilizou-se o filtro estendido de Kalman, pois este suporta uma dinâmica não linear, o que contribuirá para uma melhor estimação dos parâmetros na região ao redor da velocidade angular nula. Esse filtro é capaz de gerar trajetórias de referência que são atualizadas a cada processamento das medidas do instante correspondente (Kuga, 2005).

A aquisição dos dados foi feita em 10 Hz garantindo assim um conjunto de dados satisfatório para a estimação. A amostragem poderia ter sido feita em uma taxa mais elevada, porém o software desenvolvido em Labview apresentou alguns erros nas medidas, e o sincronismo da placa com a interface serial USB do computador ficou comprometida com taxas maiores.

Neste ensaio foram coletadas 6700 amostras em um período de 670 segundos com 4 ciclos de comando na referência da velocidade angular e amplitude de 4000 rpm. A sintonia do PI foi feita manualmente com ganho proporcional de 500 e ganho integral de 10000, porém o ganho derivativo foi mantido em zero.

Todas as amostras de corrente elétrica e velocidade de rotação foram inseridas no filtro de Kalman em Matlab, de modo a estimar a corrente elétrica, a velocidade angular, a constante do motor, o atrito viscoso e o atrito de Coulomb (Åström, 1998).

5.1.1. Modelo matemático

Uma roda de reação pode ser modelada por meio de sua inércia J_w, o atrito viscoso b e um atrito de Coulomb c conforme a seguinte equação diferencial (Carrara e Milani, 2007):

$$T_m = J_w \dot{\omega}_w + b\,\omega_w + c\,\mathrm{sgn}(\omega_w)\,, \tag{5.1}$$

onde ω_w é a velocidade angular da roda de reação e T_m é o torque do motor. Admitindo a ausência de não-linearidades na conversão eletro-mecânica do motor, pode-se considerar linear a relação entre corrente de excitação I e torque T_m:

$$T_m = k_m I\,, \tag{5.2}$$

na qual k_m é a constante do motor. Se for considerado o efeito do atrito de Stribeck segundo o modelo de Gauss (Lantos e Márton, 2010), a Equação 1 pode ser reescrita na forma:

$$T_m = J_w \dot{\omega}_w + b\,\omega_w + c\,\mathrm{sgn}(\omega_w) + (s-c)\exp(-\omega_w/\omega_s)^2\,\mathrm{sgn}(\omega_w)\,, \tag{5.3}$$

onde s e ω_s[1] são respectivamente o torque de atrito e a velocidade de Stribeck. Por fim, sob a ótica do experimento tem-se o modelo em termos de sua entrada I e saída ω:

$$I = \left[J_w\,\dot{\omega}_w + b\,\omega_w + c\,\mathrm{sgn}(\omega_w) + (s-c)\,\exp(-\omega_w/\omega_s)^2\,\mathrm{sgn}(\omega_w)\right]/k_m \tag{5.4}$$

5.1.2. Espaço Estado

Dado o problema de estimação dos parâmetros do modelo, a representação em espaço de estados será:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t, u) + \mathbf{G}\,\mathbf{w}\,, \tag{5.5}$$

sendo G a matriz de covariância e w sendo o ruído na planta. Tendo o vetor de observações dado por

$$\mathbf{y} = \mathbf{H}_k\,\mathbf{x}_k + \mathbf{v}_k\,, \tag{5.6}$$

onde k é a iteração no momento, H é a matriz de observações e v o ruído das medidas. O estado a ser estimado é

$$\mathbf{x} \equiv \begin{bmatrix} I_k & J_w/k_m & b/k_m & c/k_m & (s-c)/k_m \end{bmatrix}^T \equiv \begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \end{pmatrix}^T, \tag{5.7}$$

cujo sinal de controle fica

$$u_k \equiv (\omega_k), \tag{5.8}$$

onde ω a velocidade angular. A matriz de observações é dada por

$$\mathbf{H}_k \equiv \begin{pmatrix} I_{w_k} & 0 & 0 & 0 & 0 \end{pmatrix} \tag{5.9}$$

5.1.3. O filtro de Kalman Estendido

[1] Fator utilizado para ajuste da atenuação do termo relacionado a *s*; valores empíricos são atribuídos a ele (Romano, 2010).

O filtro de Kalman selecionado para problema é o tipo estendido, pois este suporta uma dinâmica não linear, o que certamente contribuirá para se ter uma melhor estimação dos parâmetros na região ao redor da velocidade angular nula. Esse filtro é capaz de gerar trajetórias de referência que são atualizadas a cada processamento das medidas do instante correspondente(Kuga, 2005). Para aplicação do filtro estendido são geradas as matrizes do jacobiano referentes às funções não lineares da dinâmica e das medidas.

Utilizando as equações e modelos implementados em Matlab, aplicou-se o algoritmo do filtro de Kalmam aos dados coletados. Na Fig. 5.4 mostra que nas regiões de baixa rotação ocorre um problema de rastreamento do sinal de referência do controle, onde estão presentes o atrito não linear e a baixa resolução na medição da velocidade angular. Este problema de resolução na leitura da velocidade angular poderia ser minimizado por meio de encoders de maior resolução, como mencionado no capítulo anterior, ao invés de calcular-se a velocidade pelos sensores hall.

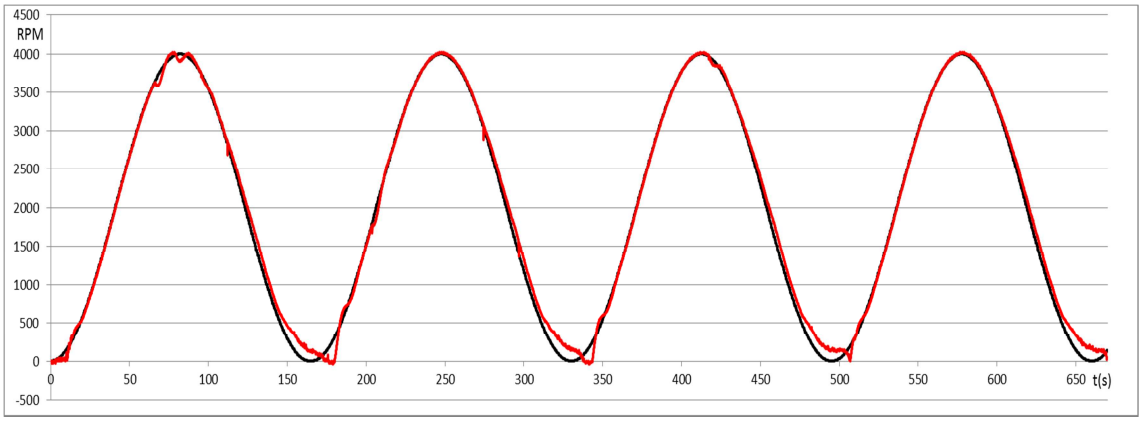


Figura 5.4 - Velocidade angular do motor (vermelho) e velocidade comandada (preto).

Observa-se, na Fig. 5.5 que o ganho de Kalman permanece próximo a zero, o que demonstra que a estimação dos parâmetros convergiram para um valor estável. De fato, quanto menor for o ganho de Kalman, mais próximo o valor estimado estará do seu valor real, embora não se possa garantir que tenha convergido para o valor correto. Ainda na Fig. 5.6, percebe-se que a constante do motor estabiliza-se num valor próximo a 41,2 mNm/A após 6700 amostras.

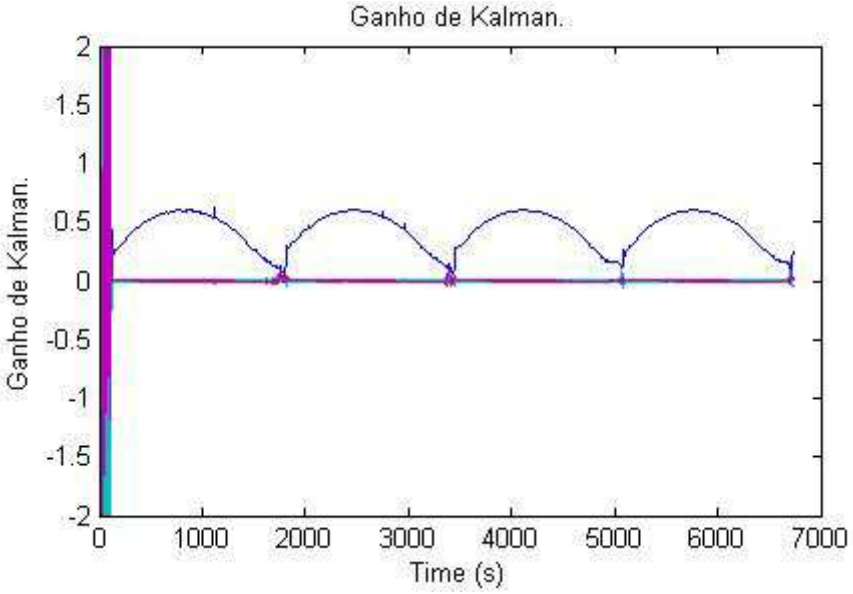


Figura 5.5 - Ganho de Kalman

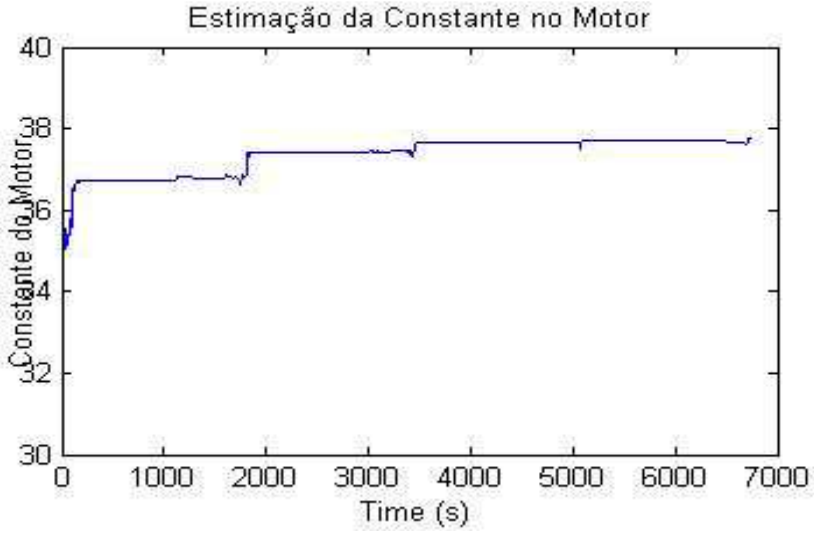


Figura 5.6 – Estimação da constante do motor

Completando os parâmetros estimados, a Fig. 5.7 mostra os resultados da estimação do coeficiente de atrito viscoso, b, e na Fig. 5.8 tem-se a estimação do coeficiente do atrito de Coulomb, c.

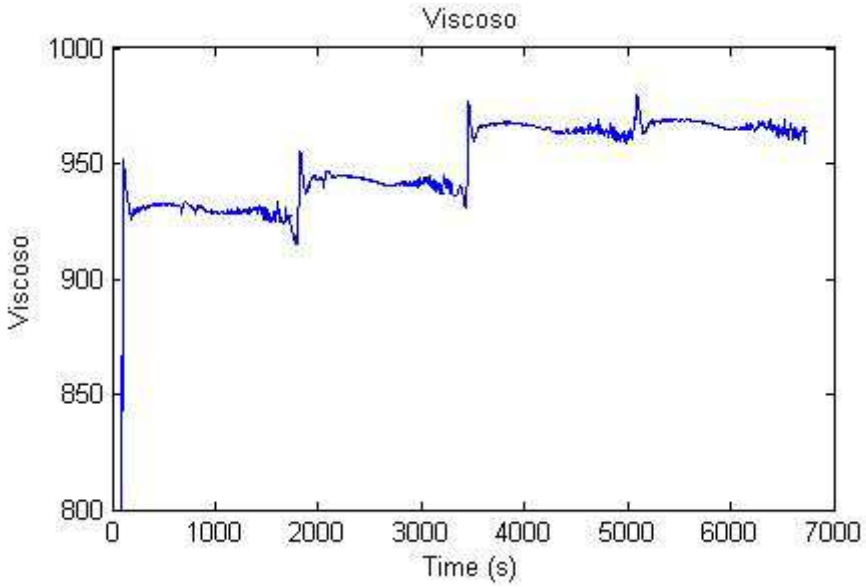


Figura 5.7 –Coeficientes de atrito viscoso

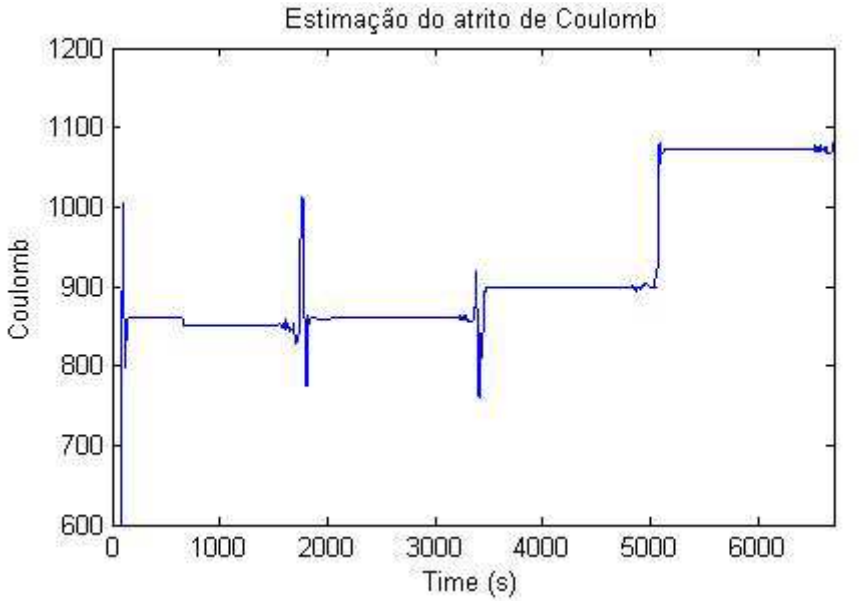


Figura 5.8 – Coeficiente de atrito de Coulomb

Estimou-se o valor de 110 $\times 10^{-3}$ Nms para o coeficiente de atrito viscoso, e 1100×10^{-3} Nm para o coeficiente de atrito de Coulomb.

Este ensaio apresentou a estimação dos parâmetros de motores BLDC (Brushless DC Motor) por meio da aplicação do filtro de Kalman no perfil de resposta do motor a uma referência em velocidade angular que varia de forma

senoidal, comandada por um controlador PI embarcado em um micro-controlador, usado como controlador do motor BLDC.

Os resultados mostraram que os parâmetros estimados variam pouco com a velocidade angular, e apresentam mudanças mais acentuadas em baixas rotações. Isto sugere que o modelo do atrito necessita ser aprimorado, pois seu efeito é mais acentuado quando o rotor se encontra em baixas velocidades. Além disso, um projeto mecânico do mancal do volante de inércia é muito importante para um ensaio mais realista condizente com uma roda de reação apta para voo,

5.2. Resposta ao degrau em velocidade

Este teste verifica a resposta da roda de reação a uma entrada na forma de um degrau. A roda foi posta a girar a (1000 rpm, e comandou-se uma velocidade de 1000 rpm por 24 segundos ao fim dos quais o comando foi novamente invertido para (1000 rpm. Neste ensaio observa-se, pela linha vermelha da Fig. 5.9, que a roda de reação partiu de (1000 rpm, no sentido anti-horário, e ao receber o comando serial para mudar a rotação para 1000 rpm, no sentido horário, cruzou o eixo horizontal (zero rpm) em 6 segundos, passando nesse momento pelo ponto de velocidade zero e atingiu um máximo de 1300rpm aos 16 segundos. Aos 24 segundos alterou-se novamente o controle para (1000 rpm no qual verifica-se o comportamento correspondente, análogo ao comando anterior.

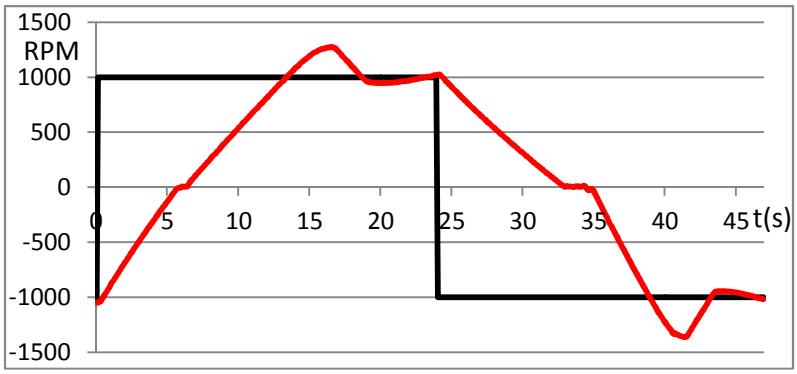


Figura 5.9 – Resposta ao degrau em 1000 rpm e -1000 rpm

5.3. Resposta ao degrau em corrente

Neste ensaio aplicou-se um patamar na corrente de referência do controle, e efetuaram-se medidas desta corrente por meio da tensão no resitor *shunt*. Primeiramente o controle foi comandado para um set-point de corrente de 100 mA. Aguardou-se a estabilização da rotação do motor, que ocorreu em 1550 rpm, com a placa atuando em controle por corrente. Após a estabilização da velocidade, aplicou-se um degrau de 100 mA no controle da corrente, passando então para 200 mA de corrente de referência. Observa-se, na Fig 5.10, em vermelho, o comportamento da corrente medida na placa, e, em preto, a corrente de referência para o controle da placa.

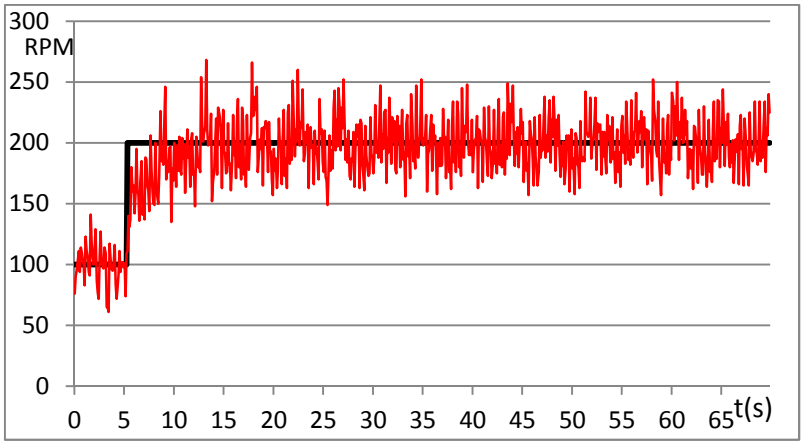


Figura 5.10 – Resposta ao degrau de corrente de 100mA a 200mA

Na Fig 5.11 tem-se o comportamento da velocidade de rotação da roda de reação, em vermelho, como resposta ao comando de degrau na intensidade da corrente. Percebe-se, na figura, que a velocidade ainda não se estabilizou mesmo após 65s de ensaio.

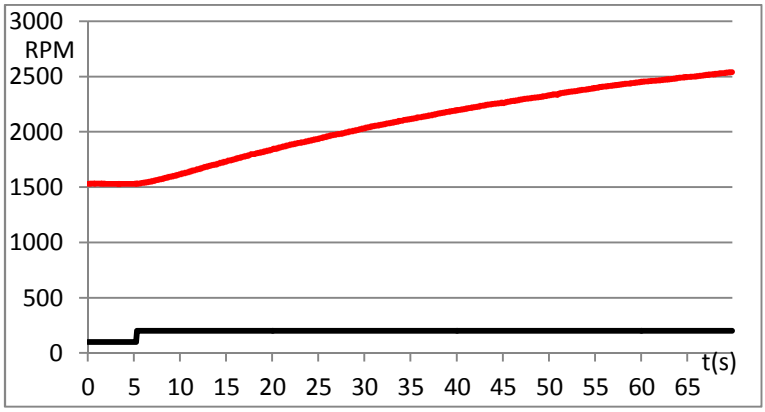


Figura 5.11 – Detalhe da rotação em resposta ao degrau de corrente de 100mA a 200mA

5.4. Tempo de decaimento em rotação

O ensaio de decaimento da velocidade de rotação foi realizado controlando a roda de reação em velocidade até sua estabilidade em cerca de 2000 rpm. Após a estabilização da velocidade, desligou-se a alimentação do motor e monitorou-se a velocidade do motor até que a roda parasse de girar, o que aconteceu após 144 segundos, como pode ser visualizado na Fig 5.12. O comportamento típico decorrente de um atrito viscoso (decaimento exponencial) aliado a um atrito seco (derivada não nula da velocidade angular no ponto de parada) é facilmente identificado nesta figura.

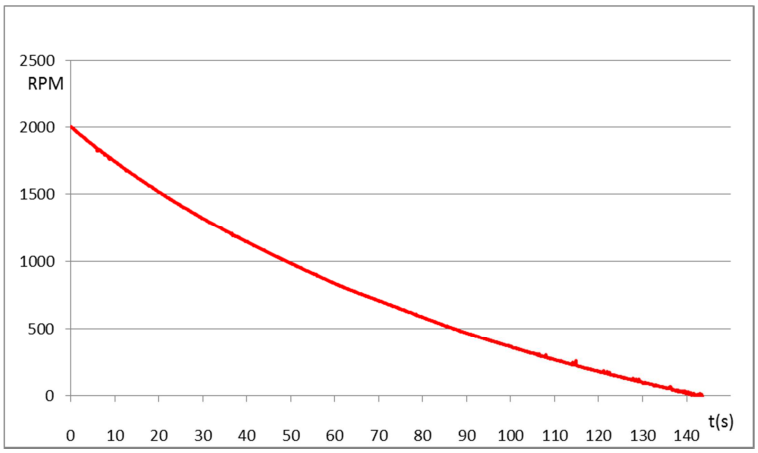


Figura 5.12 – Tempo de decaimento da roda de reação.

5.5. Estabilidade em rotação

A estabilidade do controle em velocidade angular foi medida em um ensaio no qual a roda foi posta a girar com velocidade constante. A Fig 5.13 apresenta um trecho dos dados coletados após a roda ter atingido o patamar de controle, na qual pode-se observar que a maior variação em regime foi de 15 rpm para mais e 17 rpm para menos, resultando uma variação de 0,75% para mais e 0,85% para menos respectivamente.

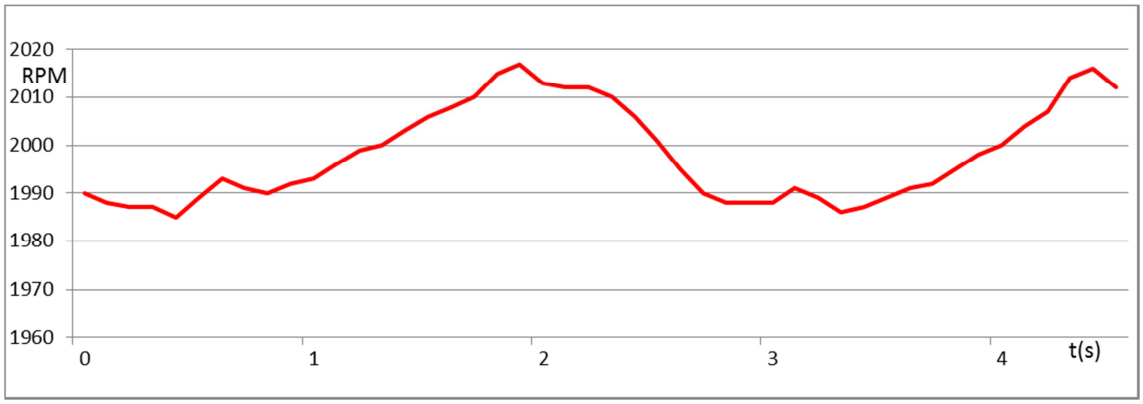


Figura 5.13 – Detalhe da leitura de rotação em regime estável

5.6. Estabilidade em corrente

Na Fig. 5.14 tem-se o gráfico que mostra a variação da corrente em controle de malha fechada por corrente, com set-point em 200 mA. A corrente medida apresentou grande variação neste tipo de controle (50 mA), o que significa que se deve melhorar a sintonia do controle PI ou mesmo projetar um modo de acionamento eletrônico mais elaborado, como por exemplo um circuito chopper, que é um controle utilizado para substituir uma tensão contínua fixa por uma tensão contínua ajustável, sendo aplicado em reguladores de tensão contínua para indutores de modo para gerar uma fonte de corrente contínua (RASHID, 2001).

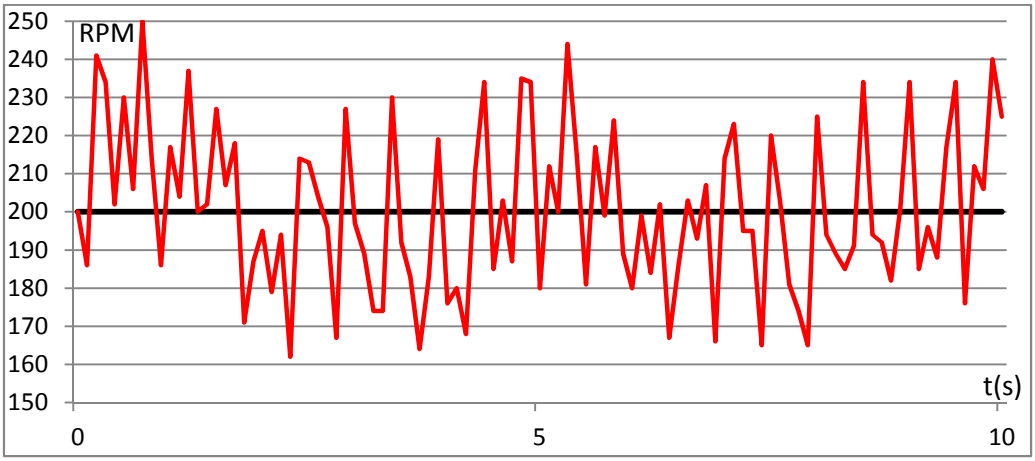


Figura 5.14 – Estabilidade no controle de corrente a 200mA.

5.7. Defasagem da rotação e magnitude da rotação

Neste ensaio procurou-se observar como a roda de reação se comporta mediante uma variação da frequência da referência da velocidade angular no controle em velocidade de rotação. Para uma baixa frequência de comando, como mostrado nas Fig. 5.15 e 5.16, não se percebe uma perda da amplitude de rotação da roda nem uma defasagem significativa do sinal comandado. Isto significa que, nesta situação, a atenuação da resposta é quase nula (1 db) e o ângulo de fase é também nulo.

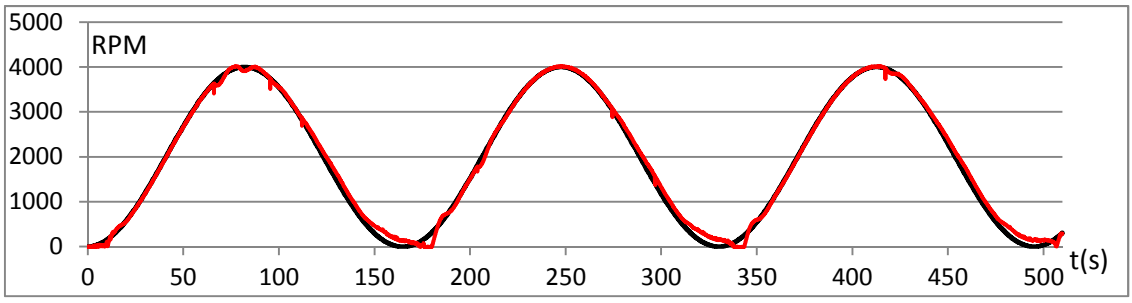


Figura 5.15 – Resposta ao comando em controle de velocidade com amplitude de 4000 rpm e frequência de 0,0055Hz.

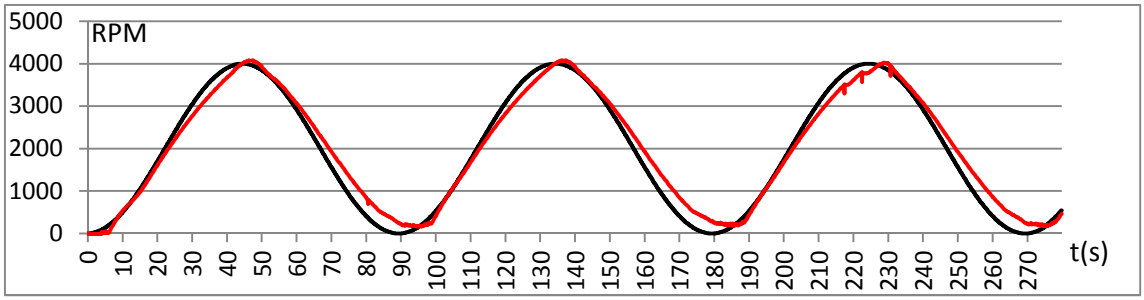


Figura 5.16 – Resposta ao comando de RPM a 0,011Hz.

Já nas Fig 5.17 e 5.18, nas quais se utilizou uma frequência no sinal de referência de 0,022 Hz e 0,06 Hz respectivamente, observa-se uma queda significativa na amplitude da rotação e um atraso entre a referência e a resposta da roda.

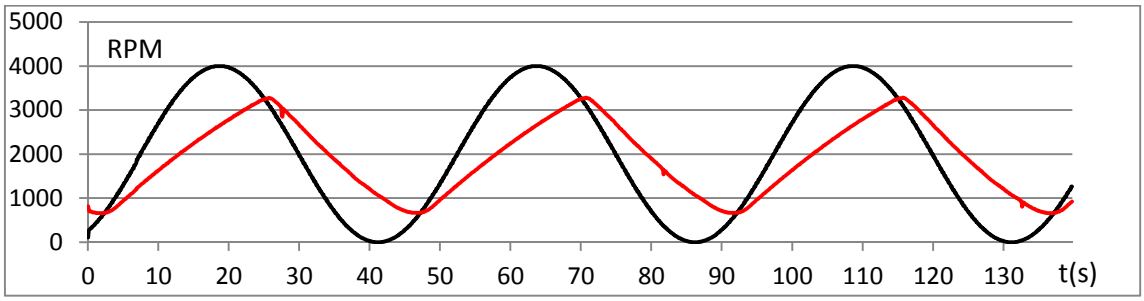


Figura 5.17 – Resposta ao comando de RPM a 0,022Hz.

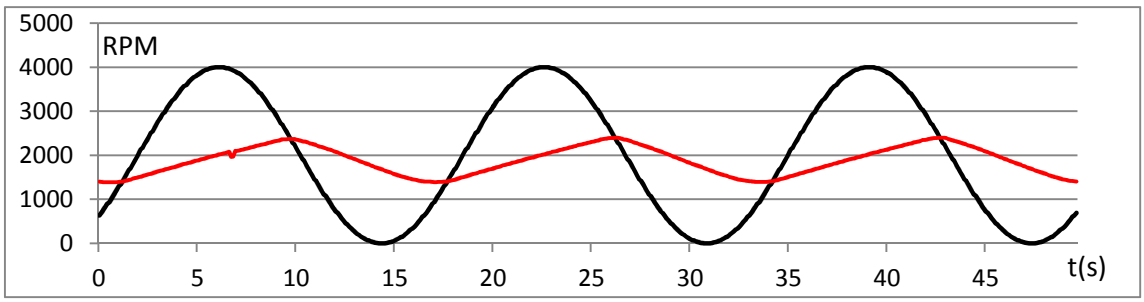


Figura 5.18 – Resposta ao comando de RPM a 0,06Hz

.

Pode-se notar na Fig. 5.19, para um comando em frequência de 0,333 Hz, que a roda já não responde ao sinal de referência, e apresenta um ângulo de fase de aproximandamente 90°.

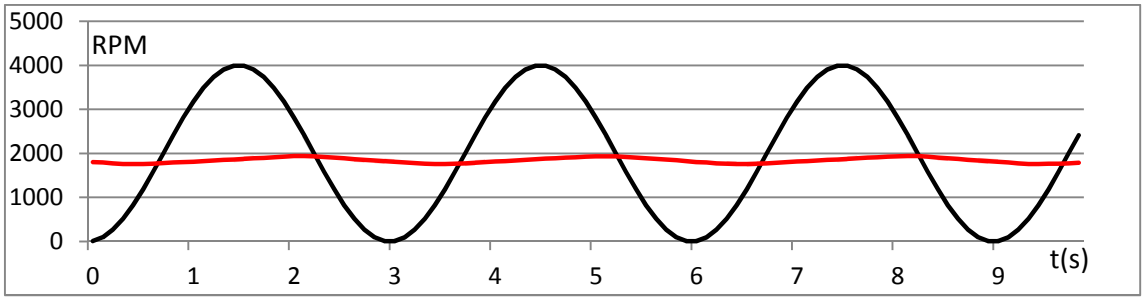


Figura 5.19 – Resposta ao comando de RPM a 0,333Hz.

5.8. Torque medido

Neste ensaio foi observada a aceleração do volante de inercia devido a aplicação de uma corrente constante no motor BLDC de valor igual a 0,72 A. Tendo o momento de inércia calculado para o volante no valor de 1,77 x 10^{-3} kg.m² e sabendo que o torque é o produto do momento de inercia pela aceleração, consegue-se calcular o torque após obter o valor da aceleração para a corrente acima mencionada.

Na Fig. 5.20 notam-se os valores da corrente medida durante os 8 segundos de ensaio, já na Figura 5.21 tem-se o acréscimo da rotação do volante de inércia.

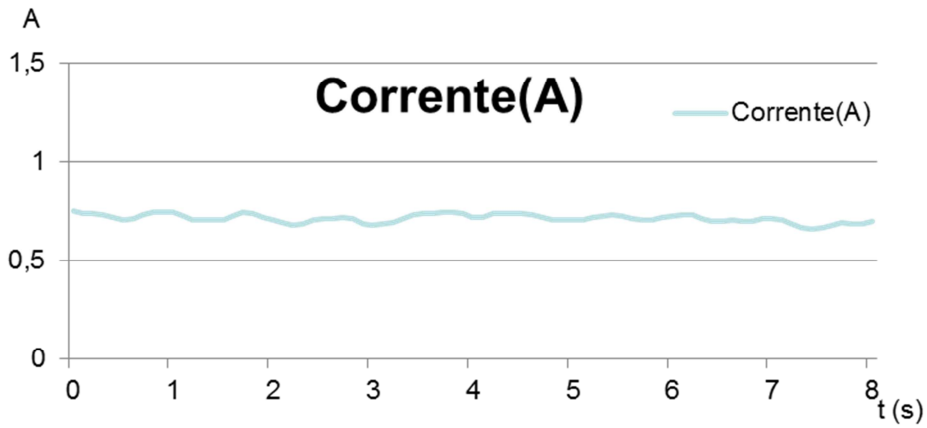


Figura 5.20 – Corrente média medida no valor de 0,72A.

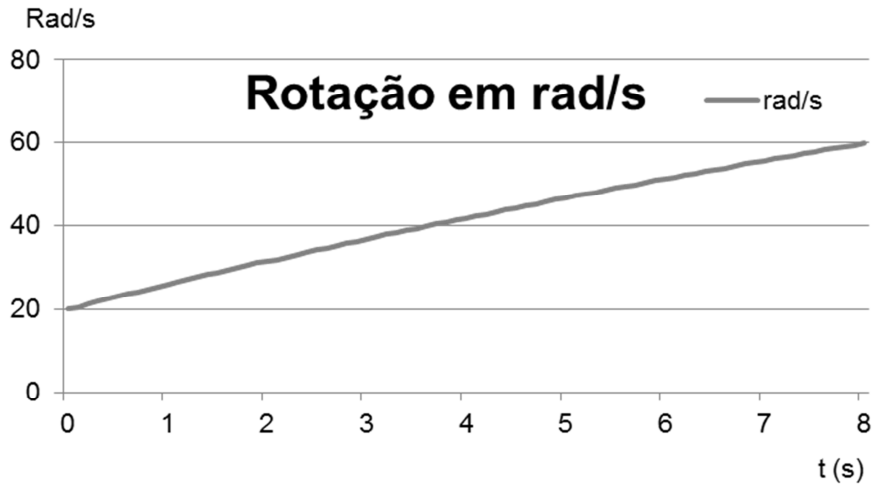


Figura 5.21 – Curva de velocidade do volante de inércia.

Com a curva de rotação medida durante o tempo de ensaio obtem-se a aceleração angular do volante de inércia como mostra a Fig. 5.22. Para o calculo do torque aplica-se:

$$T_m = JI \tag{5.10}$$

onde J é o momento de inércia do volante, I é a corrente aplicada ao motor e T_m é o torque produzido pelo motor que se deseja obter. Assim tem-se T_m igual a 10,1 x 10^{-3} Nm ou T_m / I igual a 13,2 x 10^{-3} Nm/A

O fabricante informa que o motor deve fornecer 14,1 x 10^{-3} Nm/A, mostrando desta forma que os valores medidos estão próximos aos valore de produção do motor. Na Fig. 5.22 é observada a aceleração do volante de inercia ensaiado.

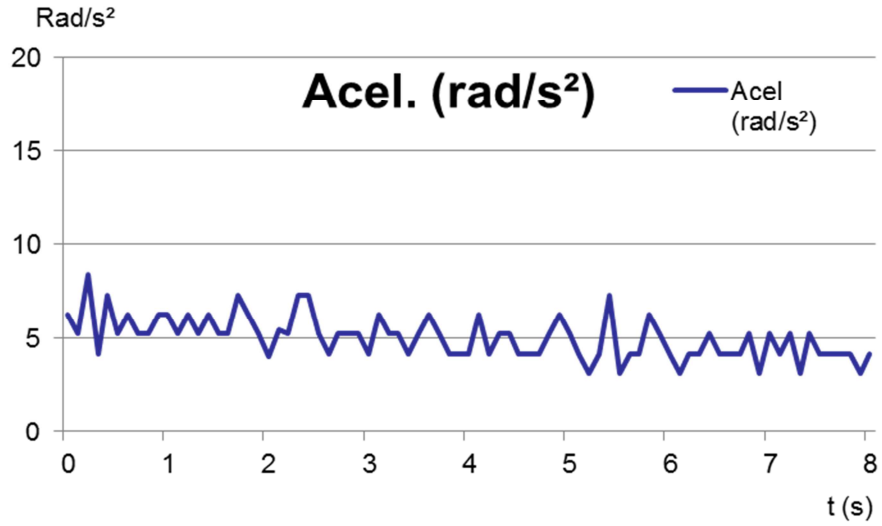


Figura 5.22 – Curva de aceleração do volante de inércia.

6. CONCLUSÕES

Este trabalho teve por objetivo o desenvolvimento de um controlador eletrônico para motores *brushless* utilizado em rodas de reação, para aplicação a possíveis experimentos de pesquisa no Instituto Nacional de Pesquisas Espaciais – INPE.

Foram apresentados o projeto do dispositivo eletrônico, a arquitetura do programa de controle no micro-controlador, incluindo um controlador PI para controle da velocidade angular do motor em malha fechada e um outro por corrente, além do circuito dos *drivers* de acionamento do motor. Foi realizado um ensaio no qual foram medidas a corrente e a velocidade angular do motor, por meio da tensão num resistor *shunt* e sensores de efeito Hall no motor, respectivamente. Os dados do ensaio foram submetidos a um filtro de Kalman, obtendo parâmetros estimados do atrito e do motor.

Os resultados mostraram que os parâmetros estimados variam pouco com a velocidade angular, e apresentam mudanças mais acentuadas em baixas rotações. Isto sugere que o modelo do atrito necessita ser aprimorado, pois seu efeito é mais acentuado quando o rotor se encontra em baixas velocidades.

Outros ensaios foram realizados de modo a caracterizar o funcionamento da roda de reação em questão e que permitirão enquadrar sua aplicação num cenário de controle de atitude em um satélite.

Com o projeto do circuito eletrônico, o desenho da placa de circuito impresso e a construção da roda de reação foi possível implementar em linguagem C o programa de acionamento das fases do motor BLDC e ensaiar a roda de reação conforme foi apresentado no capítulo anterior. Tais ensaios mostraram necessidades melhorias e correções, as quais se podem listar como melhoria:

- circuito para controle de corrente no motor (Ex.: *Chopper*);
- montagem do encoder para o disco ótico;
- algoritmo de encoder para medir rotações em baixas rotações;

- algoritmo de acionamento de fase em onda trapezoidal e senoidal.
- embutir o RTEMS no microcontrolador;
- programar a FPGA com o código do Leon e embutir o RTEMS e leis de controle nesse microntrolador, já que essa plataforma do LEON em FPGA e RTEMS deveria ser usada em diversos sensores e atuadores do INPE para missões;
- projeto mecânico do mancal do volante de inércia;

e listando os trabalhos futuros tem-se:

- estudo para substituição dos componentes por outros com Rad-hard;
- teste da roda de reação em mesa de mancal a ar;
- alteração do motor genérico por um motor capacitado para voo;
- elaboração do design mecânico para a aplicação em satélites como o Cubesat;
- estudo de ASIC Rad-hard para a roda de reação;
- desenvolvimento de tecnologia para circuito impresso Rad-hard;
- ensaios de EMI e radiação.
- usar um microcontrolador da área espacial como o ERC32, ou melhor ainda, o Leon;

Este trabalho apontou a necessidade de maiores investimentos em estudos e desenvolvimentos na tecnologia de rodas de reação, tanto no que tange ao acionamento do motor BLDC, quanto nos tipos e quantidade de sensores utilizados,

sem esquecer, ainda, de mencionar todos os problemas relativos ao mancal, aos requisitos impostos ao projeto pelo ambiente espacial e a seleção de componentes.

REFERÊNCIAS BIBLIOGRÁFICAS

ÅSTRÖM, K. J.Control of system with friction. In: INTERNATIONAL CONFERENCE ON MOTION AND VIBRATION CONTROL, 4., 1998, Zürich. **Proceedings…** Zürich: Institute of Robotics, 1998. p. 25-32.

ÅSTRÖM, K. J.; HÄGGLUND, T. **Pid controllers, theory of design and tuning**. 2. ed. Triangle Park, NC: Instrument Society of America, 1995.

CARRARA, V.; SIQUEIRA, R.; OLIVEIRA, D.Speed and current mode strategy comparison in satellite attitude control with reaction wheels. In: BRAZILIAN CONGRESS OF MECHANICAL ENGINEERING (COBEM), 21., 2011, Natal, RN, Brazil. **Proceedings…** Natal: ABCM, 2011.

CHALLAPALLI, R.; GUPTA, U. Design of a position loop servo controller for a BLDC motor based rotary electromechanical actuation system. Ranga Reddy District, India: Department of Electronics and Instrumentation, CVR College, 2008.

FONSECA, J.B.S. Estudo da aplicação de rodas de reação no sistema de controle de satélites. In: SEMINÁRIO DE INICIAÇÃO CIENTÍFICA DO INPE (SICINPE), 2011, São José dos Campos. **Anais...** São José dos Campos: INPE, 2011. p. 47. CD-ROM; Papel; On-line. Disponível em: <http://urlib.net/8JMKD3MGP7W/3AFL88E>. Acesso em: 29 jul. 2014. KUGA, H.K.**Noções práticas de técnicas de estimação.** São José dos Campos: INPE, Brasil. Notas de aula.

MARTINS, F. A.; KUGA, H. K.; CARRARA, V.; ROMANO, R. A. Estimação de parâmetros de um motor DC sem escovas usando filtro de Kalman. In: CONGRESSO NACIONAL DE ENGENHARIA MECÂNICA, 7. (CONEM), 2012, Sao Luis Rio de Janeiro. **Anais...** 2012. p. 1-7. DVD..

MAXON Motor. 2014. Disponível em: www.maxonmotor.com/medias/ sys_master/ 8813633929246/14-225-en.pdf

RASHID, M. H. **Power electronic handbook**. Academic Press, 2001. p 78 - 98. 0125816502, 9780125816502.

ROMANO, R. A. **Identificação de processos não-lineares e quantificação de atrito em válvulas de controle**.Tese (Doutorado em Engenharia de Sistemas) – Escola Politécnica da Universidade de São Paulo, São Paulo, Brasil, 2010.

SANTANA, A.C. **Controle de atitude de satélites artificiais**. 2008. Monografia (Graduação em Engenharia de Controle e Automação) - U.F. Ouro Preto, Brasil, 2008.

SOUZA, P.N.; FLEURY, A.T. Modelo experimental de uma roda de reação para controle da atitude de satélites artificiais: construção, simulação e testes. In: CONGRESSO BRASILEIRO DE ENGENHARIA MECÂNICA, 9., (COBEM), Florianópolis. **Anais...** ABCM, 1987. (INPE-4274- PRE/1146).

SOUZA, P.N.**Análise do mancal do Experimento Roda de Reação: determinação de suas frequências naturais de vibração e das cargas atuantes durante os testes de vibração**. São José dos Campos: INPE, 1994. 43 p. (A-ETD-0084).

SOUZA, P.N. Uma metodologia para a determinação da confiabilidade de mancais de rolamento sujeitos a variações de carga, rotação e temperatura. In: CONGRESSO BRASILEIRO DE ENGENHARIA MECÂNICA, 12., (COBEMCIDIM/95), 1995, Belo Horizonte, **Anais...** (INPE-5656-PRE/1831).

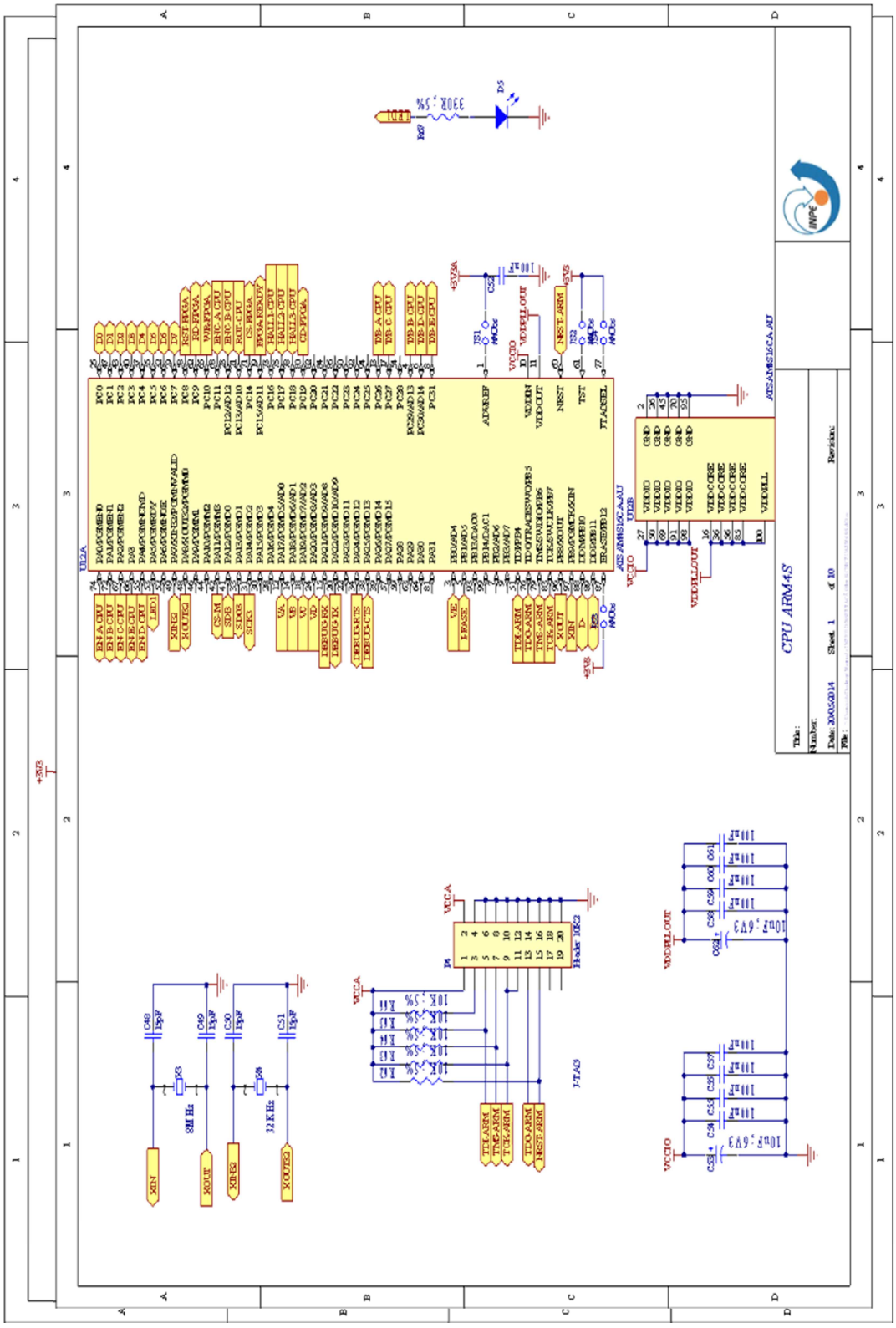
TRIVELATO, G. C. **Controle de rodas de reação através de técnicas digitais usando modelos de referência**. 1988. 209 p. (INPE-4618-TDL/335). Dissertação (Mestrado em Mecânica Espacial e Controle) - Instituto de Pesquisas Espaciais, São José dos Campos, 1988.

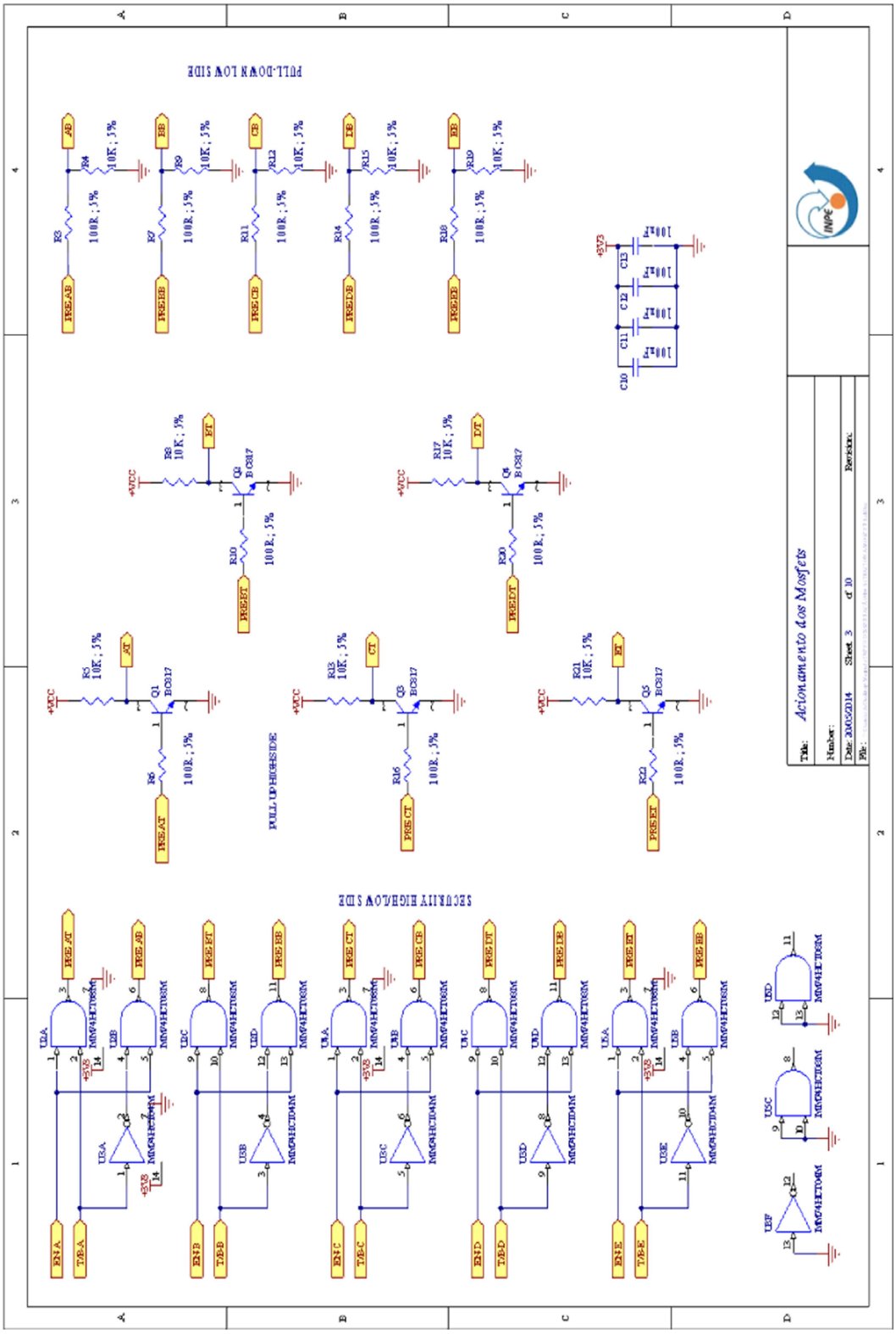
TRIVELATO, G. C.; SOUZA, M. L. O. Projeto de um controlador digital de torque de rodas de reação usando modelos de referencia. In: SBA CONGRESSO BRASILEIRO DE AUTOMATICA, 7., 1988. São José dos Campos. **Anais...** SBA, 1988. p.595-601. (INPE-4753-PRE/1425).

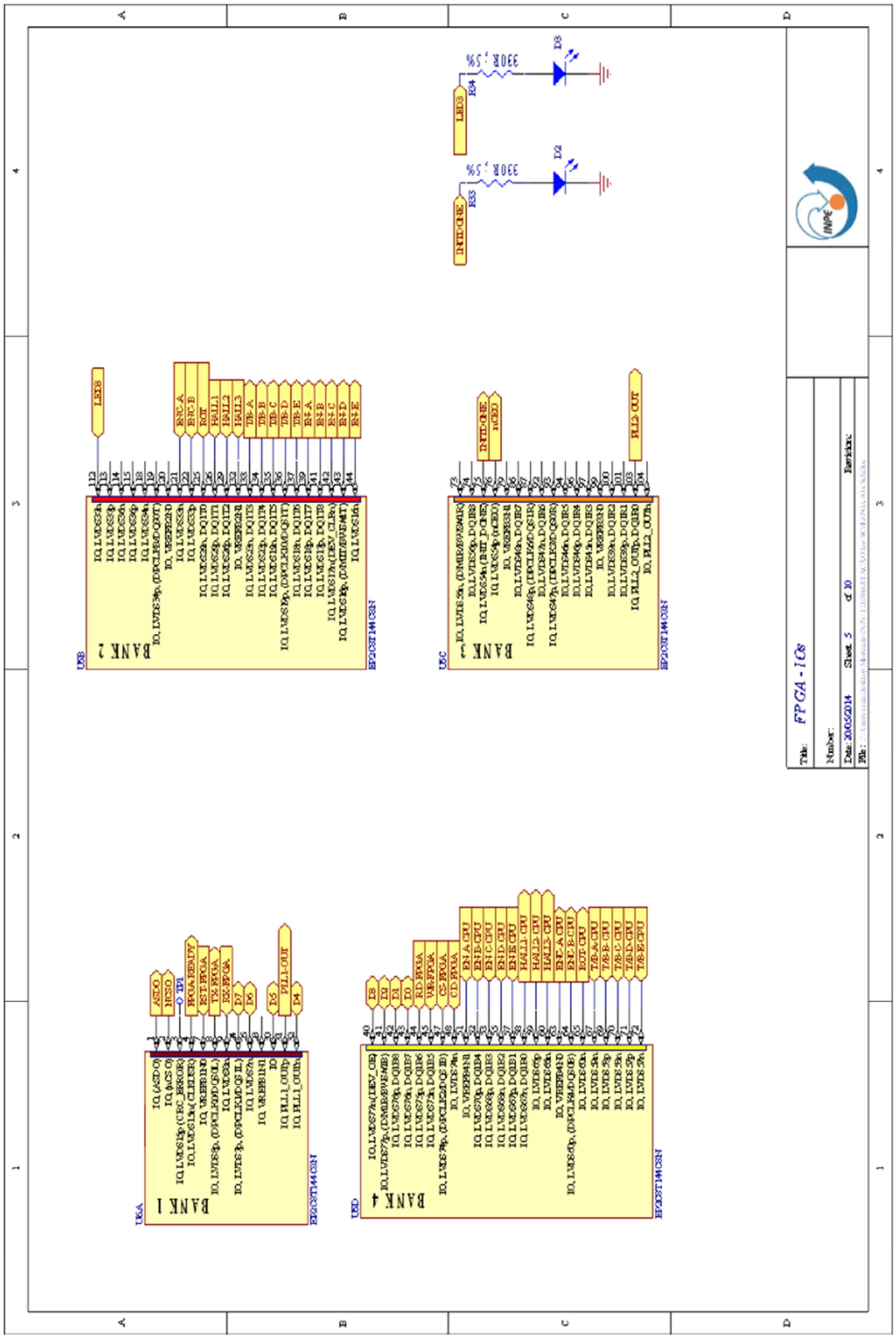
RAJAN, A. A.; VASANTHARATHNA, S. Harmonics and torque ripple minimization using LC filter for Brushless DC motors **Interantional Journal of Recent Trends in engineering**, v. 2, n. 5, Nov. 2009.

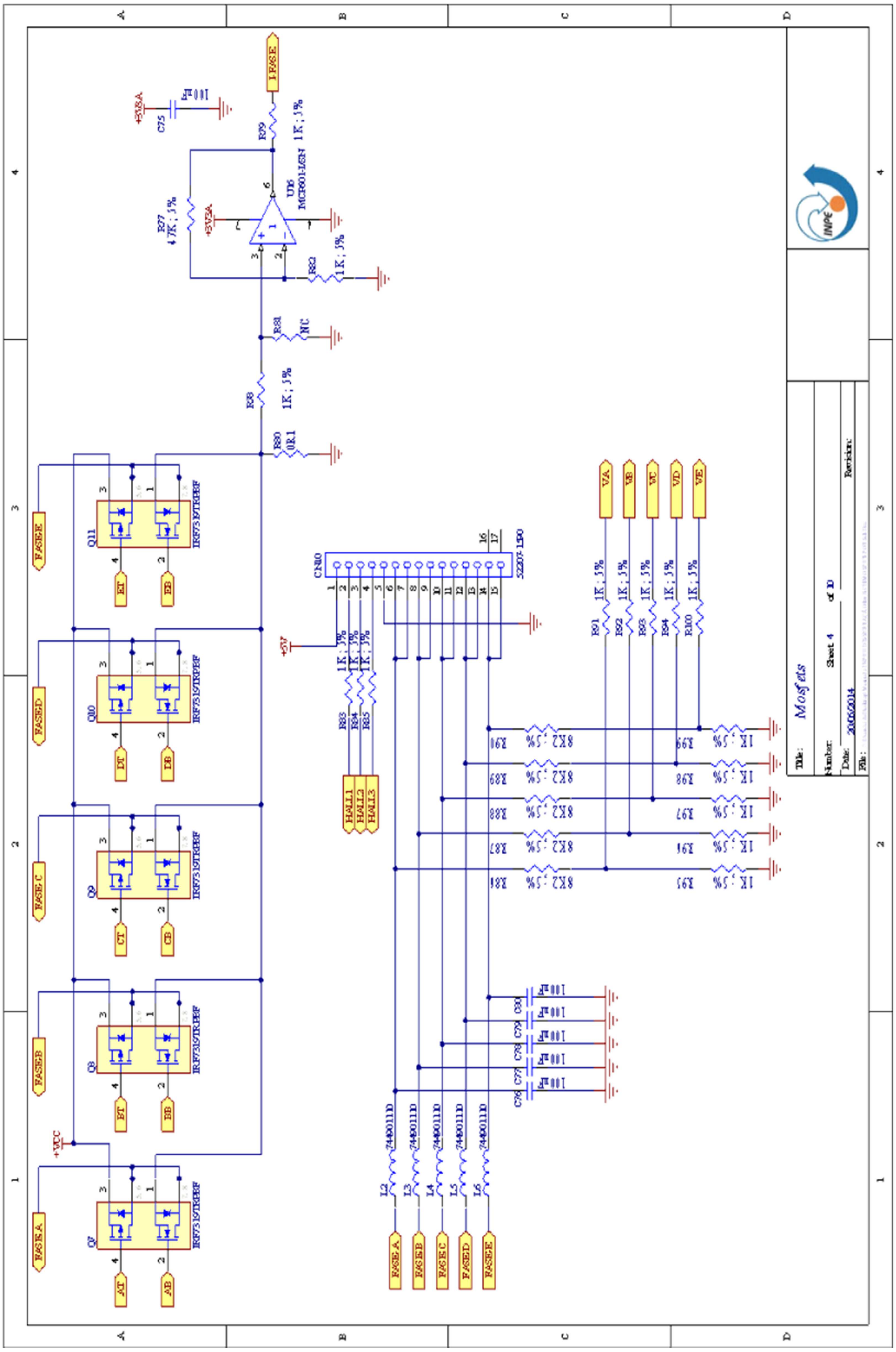
WERTZ, J.R. **Spacecraft attitude determination and control**. London, England: D. Reideil Publishing Company, 1978. ISBN:9027709599.

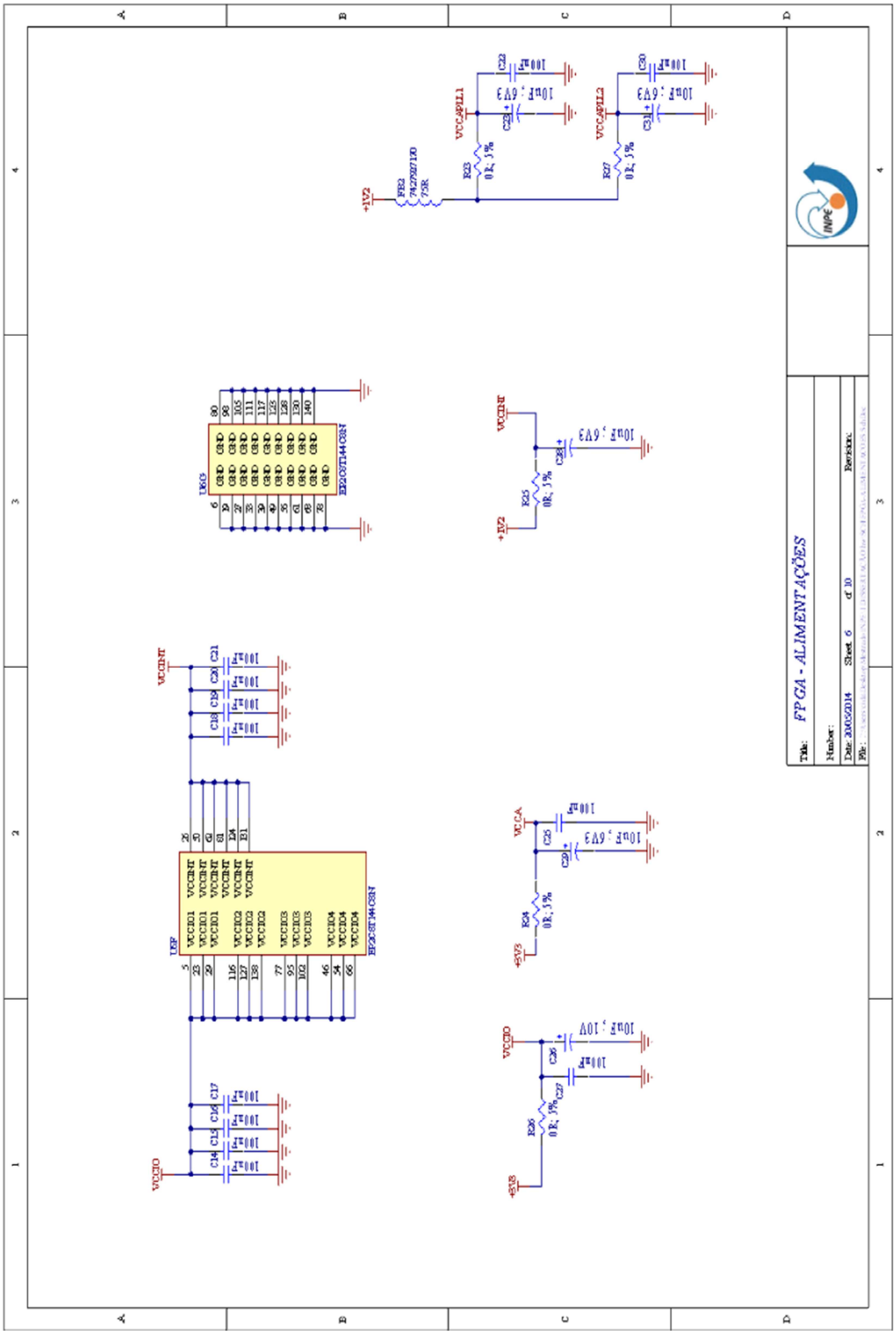
APÊNDICE A – ESQUEMA ELÉTRICO DO CONTROLADOR

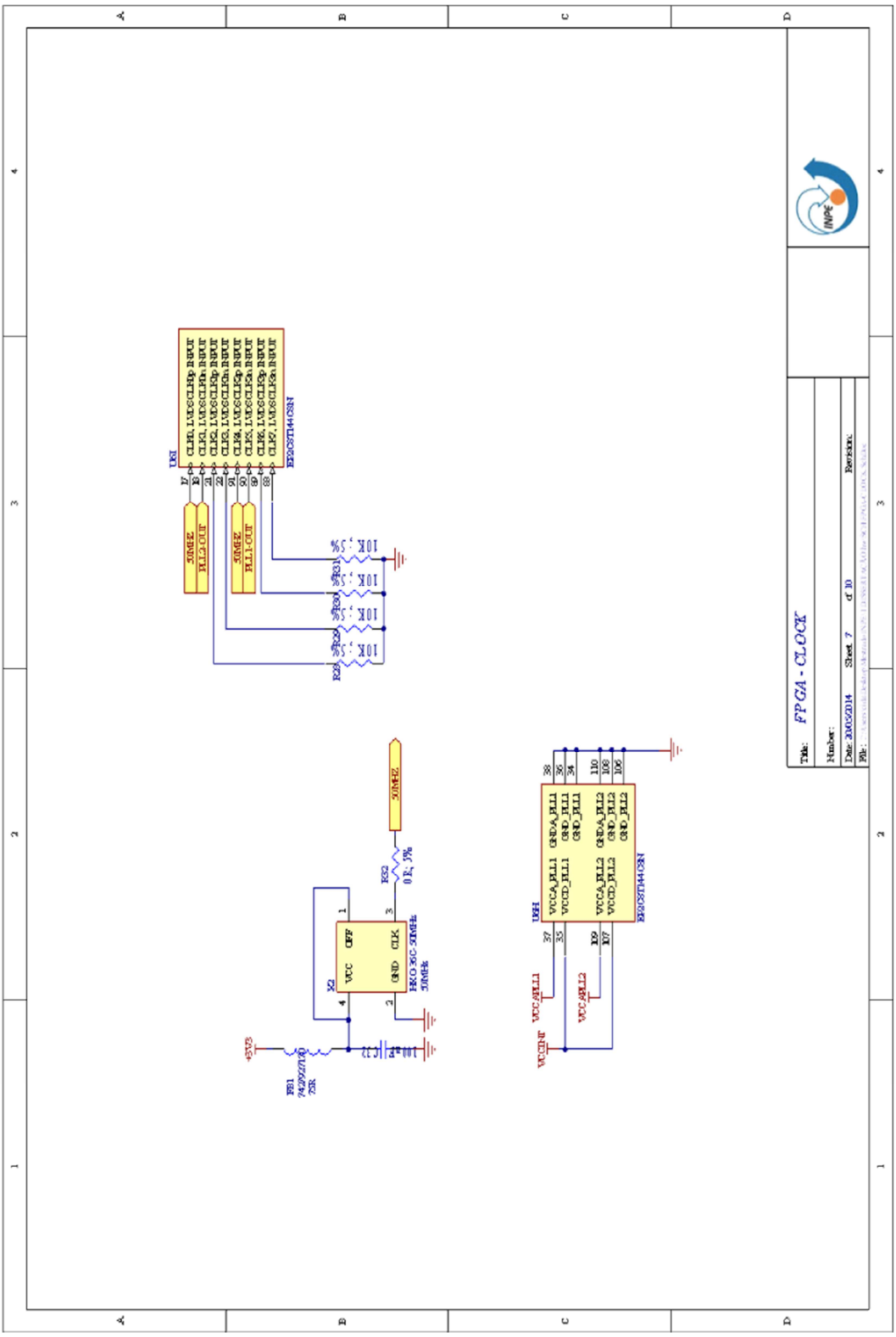


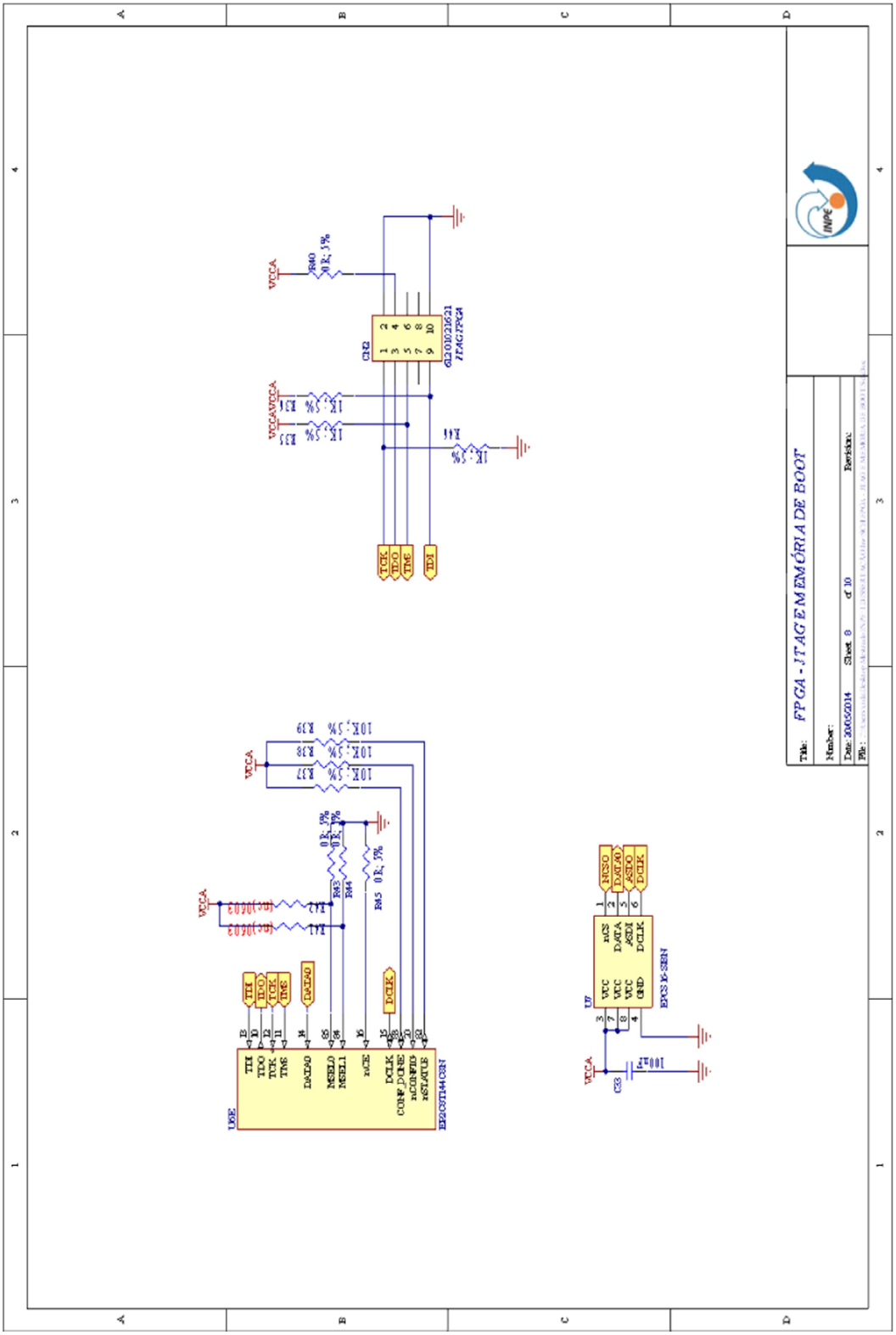

PULL-DOWN LOW SIDE
SECURITY HIGH/LOW SIDE
PULL UP HIGH SIDE
Title: Acionamento dos Mosfets
Number:
Date: 20/05/2014 Sheet 3 of 10 Revision:
File:

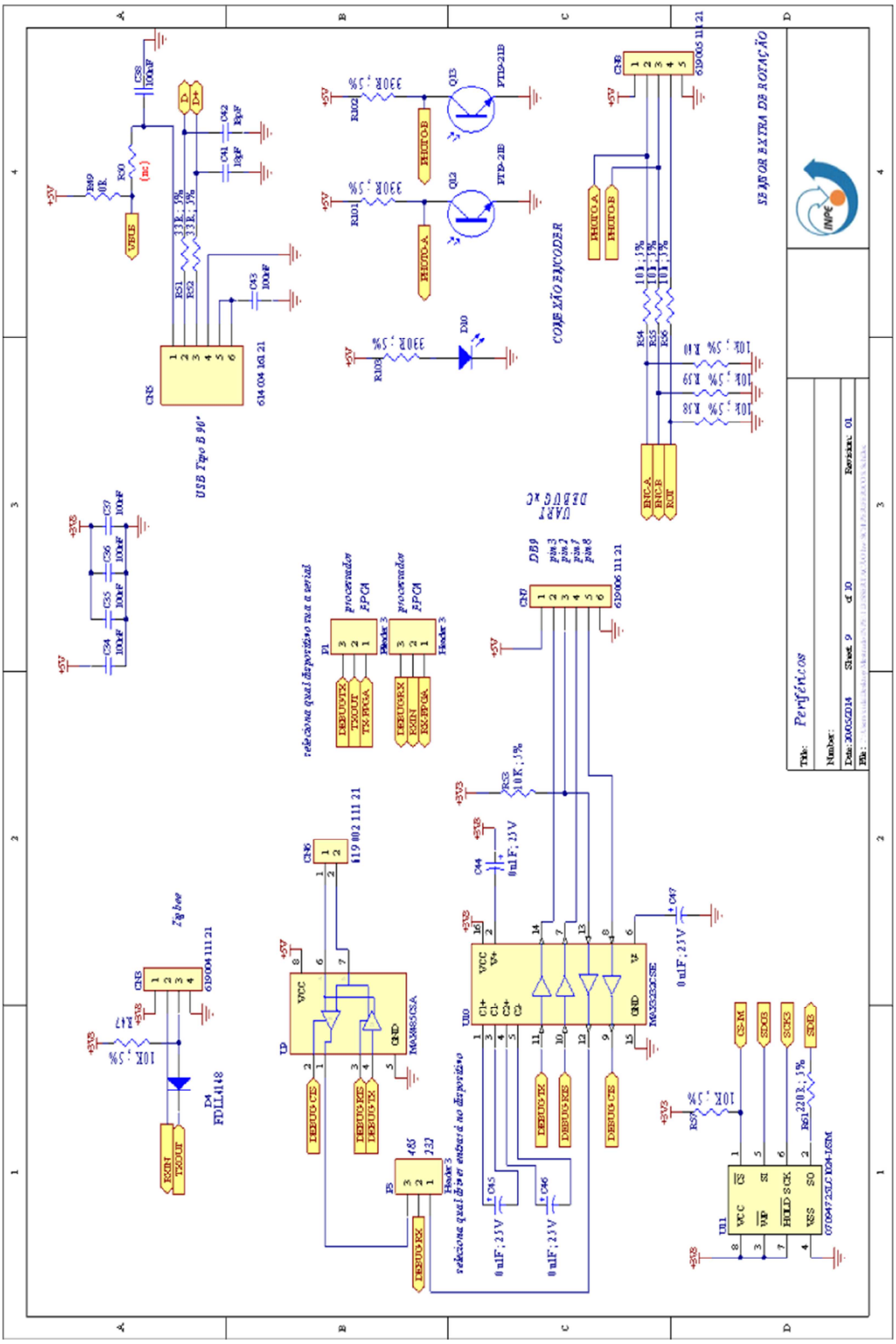

FPGA - 1 Cx
BANK 1
BANK 2
BANK 3
BANK 4
LED8
LED3
LED2
330R, 5%
INTERRUPT
PLL2_OUT
PLL1_OUT
EP2S30F1440I3N

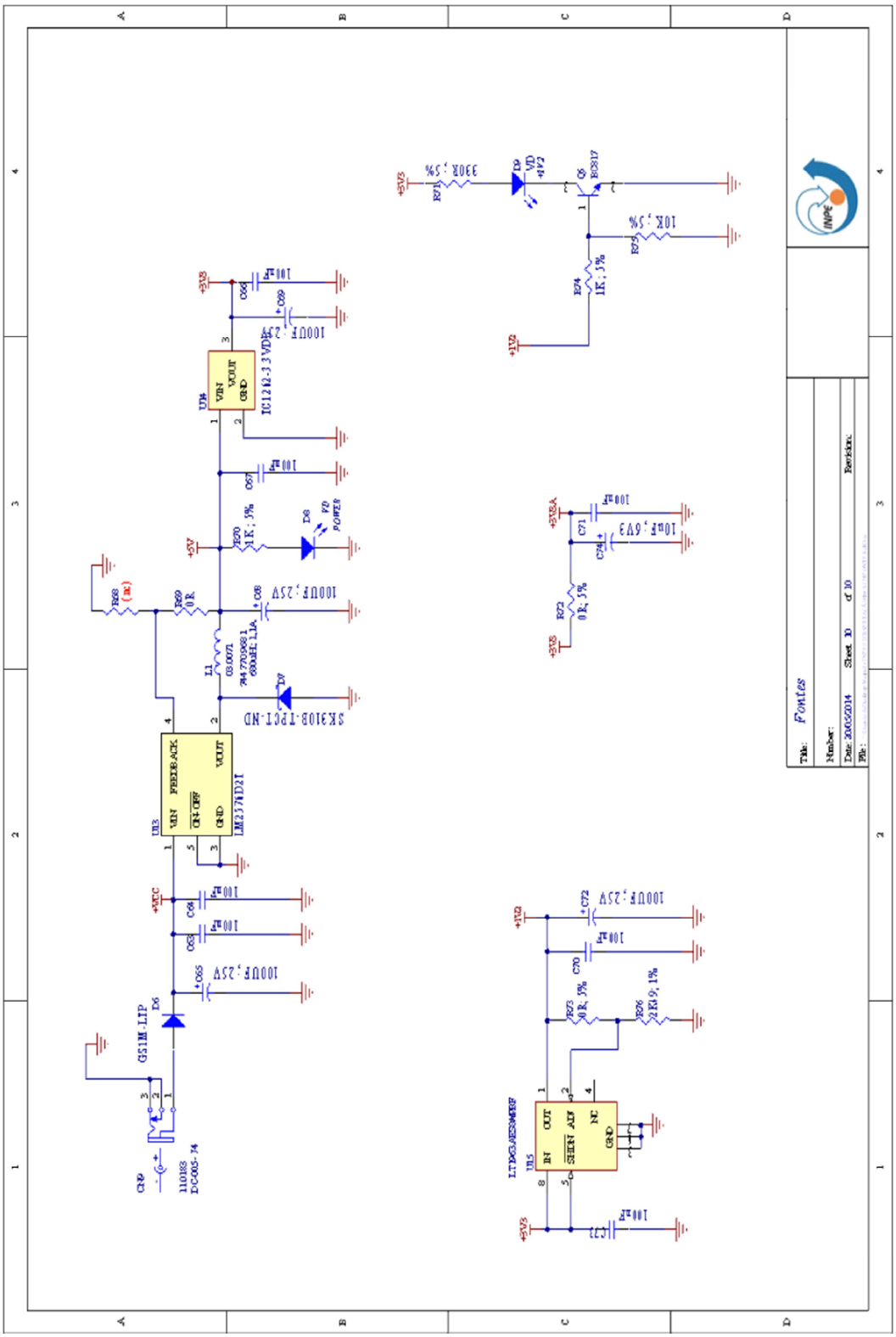

Mosfets
Sheet 4 of 10
INPE


FPGA - ALIMENTAÇÕES
Title:
Number:
Revision:
Date: 20/05/2014
Sheet 6 of 10
File:
INPE
+1V2
+3V3
VCCINT
VCCIO
VCCA
VCCAPLL1
VCCAPLL2
FB2
742792100
75R
R23
0R, 1%
R27
0R, 1%
R25
0R, 1%
R24
0R, 1%
R26
0R, 1%
C22
100nF
C23
10uF, 6V3
C30
100nF
C31
10uF, 6V3
C28
10uF, 6V3
C25
100nF
C29
10uF, 6V3
C36
100nF
C37
10uF, 10V
C21
100nF
C20
100nF
C19
100nF
C18
100nF
C17
100nF
C16
100nF
C15
100nF
C14
100nF
U6F
EP2C8T144C8N
U6G
EP2C8T144C8N

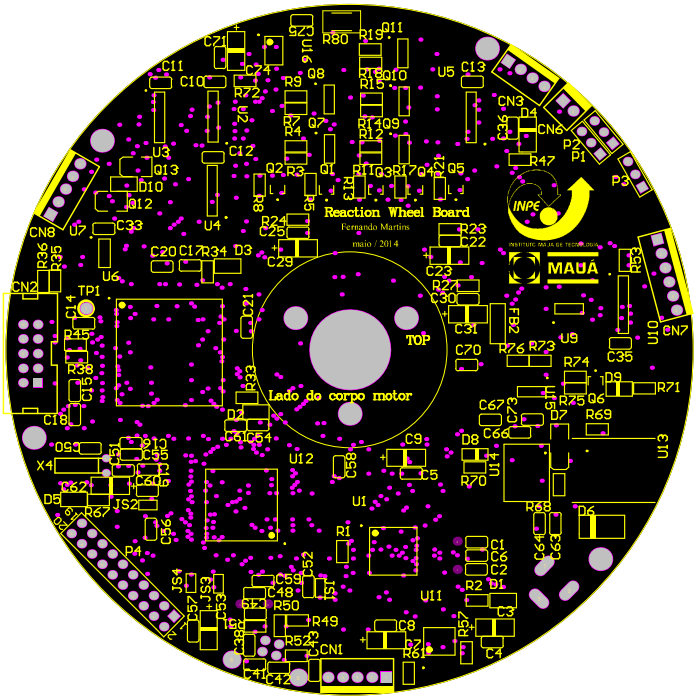

FPGA - CLOCK
Title:
Number:
Revision:
Date: 2005/2014
Sheet 7 of 10
File:
INPE
50MHZ
PLL1-OUT
PLL2-OUT
10K, 5%
0 R, 5%
VCC_PLL1
VCC_PLL2
VCCINT


FPGA - JTAG E MEMÓRIA DE BOOT
Title:
Number:
Date: 20/05/2014
Sheet 8 of 10
Revision:
File:
INPE
VCCA
VCCA/VCCA
TCK
TDO
TMS
TDI
JTAG FPGA
10K, 5%
0R, 5%
DATA0
DCLK
nCS
ASDI
nCSO
EP3C5T144C8N
EPCS16SI8N
100nF

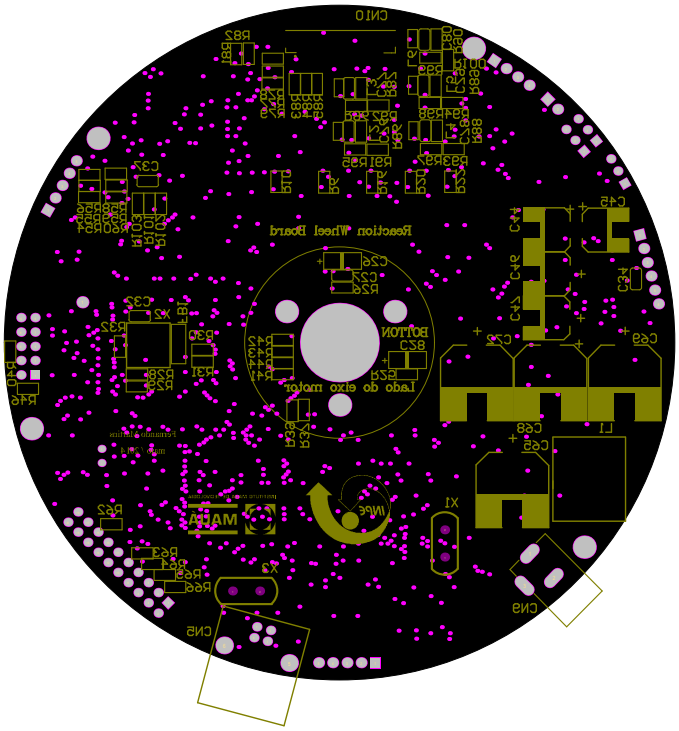

SENSOR EXTRA DE ROTAÇÃO
CONEXÃO ENCODER
USB Tipo B 90°
Zigbee
UART DEBUG
Periféricos



APÊNDICE B – Disposição dos componentes

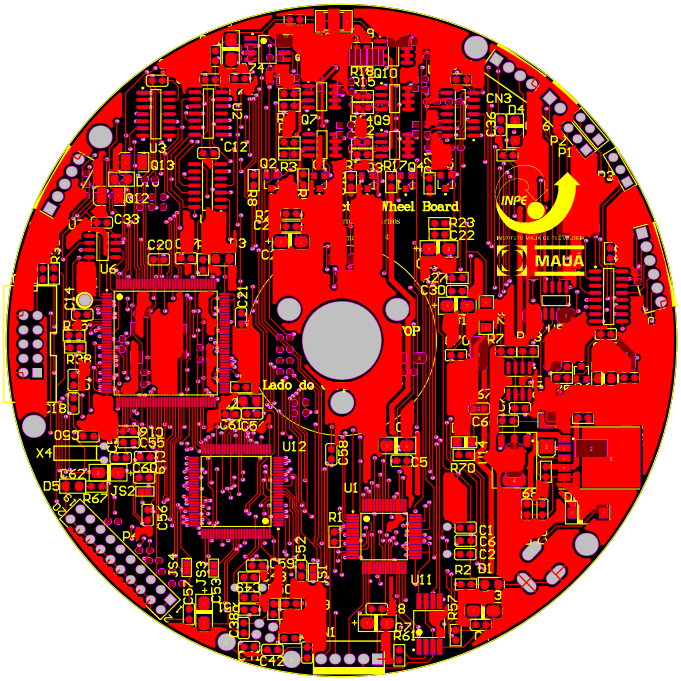


Vista superior, detalhe dos componentes

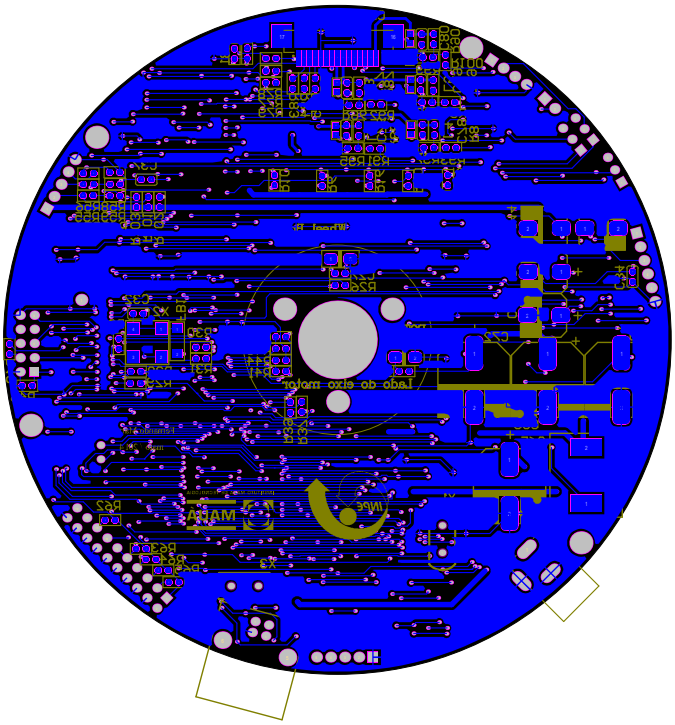


Vista inferior, detalhe dos componentes.

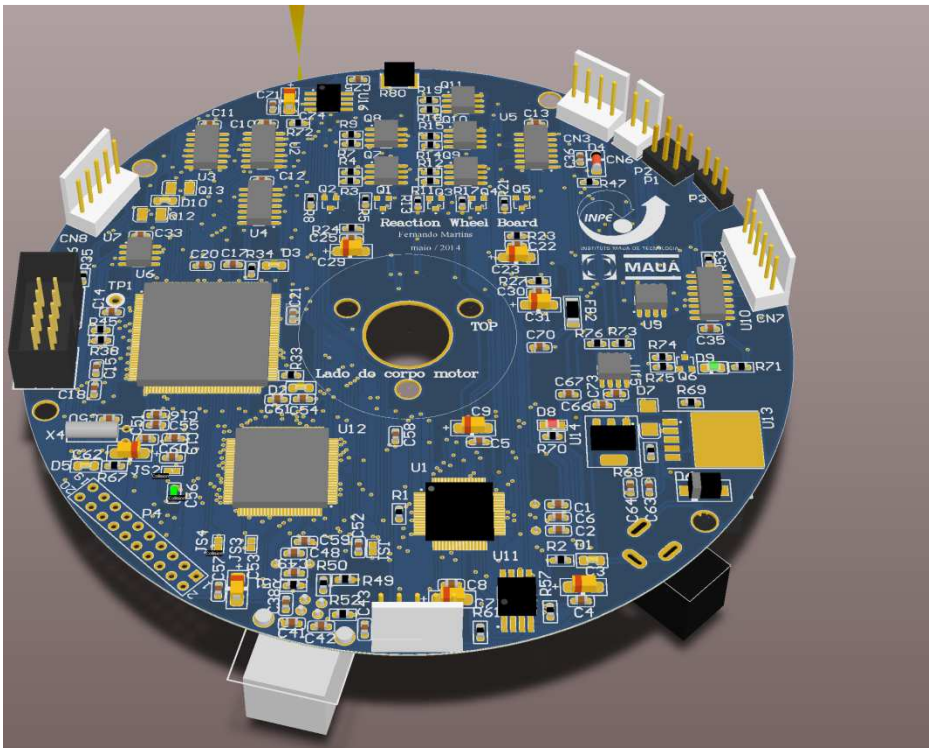
APÊNDICE C – VISTA DA PLACA DO CONTROLADOR



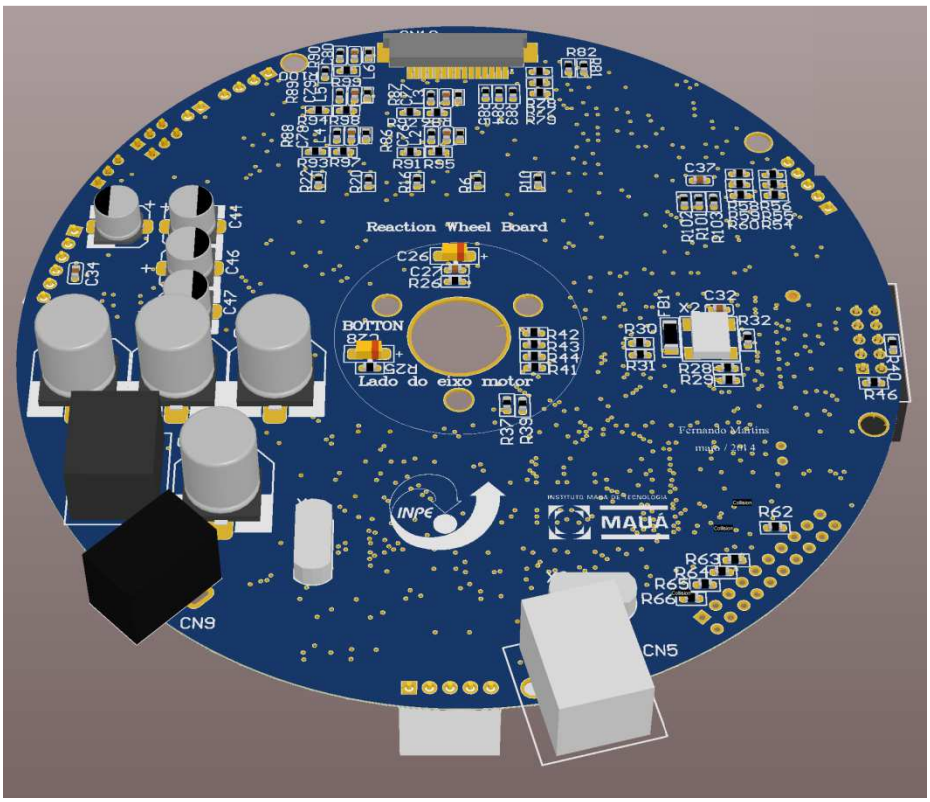
Vista superior, detalhe dos componentes



Vista inferior do layout da placa de circuito impresso



Vista superior 3D da placa de circuito impresso projetada



Vista inferior 3D da placa de circuito impresso projetada

APÊNDICE D – LISTA DE COMPONENTES

Descrição	Valor	Quant.
Capacitor Cerâmico SMD	15pF	6
Capacitor de tântalo	10uF ; 6V3	9
Capacitor Cerâmico SMD	100nF	22
Capacitor de tântalo	10uF ; 10V	2
Capacitor cerâmico	100nF	28
Capacitor cerâmico	18pF	2
Capacitor Eletrolítico SMD	0u1F ; 25V	4
Capacitor Eletrolítico SMD	100UF ; 25V	4
Conector-BOX-header	10 vias (2 x 5)	1
Conector USB - Fêmea Tipo	USB	1
Led smd		5
Diodo de Sinal	LL4148	1
Diodo retificador	LL4007	1
Diodo Schottky	SK310B-TPCT-ND	1
Indutor	75R	2
Indutor	680uH ; 1,1A	1
Indutor	10nH ; 600mA	5
NPN General Purpose Amplifier	BC817	6
HEXFET Power MOSFET	IRF7319TRPBF	5
NPN Phototransistor	PT19-21B	2
Resistor 10K ; 5%		1
Resistor 330R ; 5%		7
Resistor 1K ; 5%, 8K2 ; 5%, 10K ; 5%, 47K ; 5%, 100R ; 5%, N		50
Resistor 0R; 5%, 1K ; 5%, 1K; 5%, 2K49; 1%, 10K ; 5%, 10K; 5%, 330R ; 5%		34
Resistor 0R		5
Resistor 33R ; 5%		2
Resistor 220R ; 5%		1

Resistor 3K24 ; 0,1%		1
Resistor 0R1		1
Microcontrolador PIC	PIC32MX675F512H	1
Quad 2-Input AND Gate	MM74HCT08M	3
Hex Inverter	MM74HCT04M	1
Cyclone II Family	EP2C8T144C8N	1
EPCS16		1
Low-Power, RS-485/RS-422 Transceiver	MAX485CSA	1
3.0V TO 5.5V, Low-Power,Transceiver	MAX3232CSE	1
AT91 ARM Cortex-M4 32-bit Microcontroller,	ATSAM4S16CA-AU	1
Regulador Chaveado Tensão	LM2576	1
Regulador de tensão	TC1262-3.3	1
1.5 A LDO Regulator	LT1963AES8#PBF	1
Amplificador Operacional	MCP601	1
Cristal oscilador	8MHz	2
Oscilador	50MHz	1
Cristal oscilador	32KHz	1

APÊNDICE E – PROGRAMA DE CONTROLE

No presente apêndice é apresentado o programa de controle da roda de reação proposta, porém a parte de firmware, onde são feitos os tratamentos de configuração de periférico, não serão mostrados para não deixar o apêndice muito extenso.

```
/*********************************************************************
*******************\
 *                          Software para controle de motor BLDC
*
 *
*
 *                      Desenvolvido por Fernando de Almeida Martins
*
 *
*
 *                           E-mail: fmartins_eng@yahoo.com
*
 *
*
 * Data: 13/02/2012
Versão: 1.0 *
 *
*
\*********************************************************************
*******************/

/*********************************************************************
*******************\
 *                                      Controle de Versões
*
 *
*
 * Responsável         Versão             Data      Descrição
*
 *
*
 * Fernando Martins  sw.h               13/02/2012  Rotinas bases
*
 *  Fernando Martins  sw.h               13/05/2015 Sintonia para
protótipo         *
*
\*********************************************************************
*******************/


#ifndef     _SW_H_
#define     _SW_H_

#include "fw.h"
```

```
/*********************************************************************
*******************\
 *******
*********
 *******                                MAPAS DE FLAGS
*********
 *******
*********
\*********************************************************************
*******************/

extern union unsigned_char flags_sw1;                    // Flags de
software
extern union unsigned_char flags_sw2;                    // Flags de
software

/*********************************************************************
*******************\
 *                                    Definição de constantes
                    *
\*********************************************************************
*******************/

/************************************************\
 *              Diretivas de compilação           *
\************************************************/

// Modo de execução do software: teste de firmware-base ou release
#define SW_TESTE                                    0
#define SW_RELEASE                                  1
#define TIPO_SW                                     SW_RELEASE

/************************************************\
 *                       Flags                    *
\************************************************/

#define FS_RESERVADO                    flags_sw1.bit0           //
Indica se o código de barras deve ser zerado ou não após sair da tela
de solicitação do código de barras

/************************************************\
 *                    Auxiliares                  *
\************************************************/
typedef enum {
BLDC_PARALIZADO,
BLDC_ON,
} SM_BLDC;



#define TIMEOUT_BLDC                                              10
// 10 * 1ms
#define TIMEOUT_PROTOCOLO_AUTOMATICO                              100
// 100 * 1ms

// PID:
#define PID_SPEED                                                 0
// Comente esta linha caso não queira utilizar PID para o controle da
bomba BFP
```

```
#define PID_CURRENT                                                    1
// Comente esta linha caso não queira utilizar PID para o controle da
bomba BFP

#define PID_SPEED_N                                          20.0
#define PID_SPEED_H                                          40.0
#define PID_SPEED_TT                                         40.0
#define PID_SPEED_KP                                         500.0
//500000.0
#define PID_SPEED_KI                                         10000.0
#define PID_SPEED_KD                                         0.0

#define PID_CURRENT_N                                        20.0
#define PID_CURRENT_H                                        40.0
#define PID_CURRENT_TT                                       40.0
#define PID_CURRENT_KP                                       1000.0
#define PID_CURRENT_KI                                       10000.0
#define PID_CURRENT_KD                                       0.0



#define QTD_MEDIA_RPM                              32
#define QTD_MEDIA_CORRENTE                         128

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

#define NUM_PAR_POLOS                              4
#define NUM_FASES                                  3
#define PASSOS_COMUTACAO                           6
#define PASSOS_POR_VOLTA
NUM_PAR_POLOS*NUM_FASES*PASSOS_COMUTACAO
#define HALLS_POR_VOLTA
NUM_PAR_POLOS*NUM_FASES*PASSOS_COMUTACAO

/*********************************************************************
*******************\
 *                                                 Macros
                            *
\*********************************************************************
*******************/


/*********************************************************************
*******************\
 *                                  Definição de estruturas do módulo
                   *
\*********************************************************************
*******************/

/*********************************************************************
*******************\
 *                       Definição de variáveis do módulo em memória de
programa                 *
\*********************************************************************
*******************/

// - Globais ao sistema:
```

```
/*********************************************************************
******************\
 *                        Definição de variáveis do módulo em memória
de dados                 *
\*********************************************************************
******************/

// - Globais ao sistema:



/*********************************************************************
******************\
 *                                          Prototipagem
                              *
\*********************************************************************
******************/

inline void inicializa_sw( void );
inline void sm_processo_bldc( void );

void calcula_rpm( void );
void calcula_rpm_medio( void );
void calcula_corrente_media( void );

void formata_valor( long numero, unsigned char casas_decimais,
unsigned char opcao, unsigned char *str_valor );
void copia_string( const unsigned char *origem, unsigned char
*destino, unsigned char tamanho );
unsigned char tamanho_string( unsigned char *str );

void inicializa_memoria_dados_eeprom( void );
void carrega_dados( void );
void salva_dados( void );

/*********************************************************************
******************\
 *                                 Mensagens de error ou warning
                  *
\*********************************************************************
******************/


#endif //_SW_H_
```

```
/*********************************************************************
*******************\
 *                          Software para controle de motor BLDC
*
 *
*
 *                      Desenvolvido por Fernando de Almeida Martins
*
 *
*
 *                           E-mail: fmartins_eng@yahoo.com
*
 *
*
 * Data: 13/02/2012
Versão: 1.0 *
 *
*
\*********************************************************************
*******************/

/*********************************************************************
*******************\
 *                                       Controle de Versões
*
 *
*
 * Responsável         Versão             Data      Descrição
*
 *
*
 * Fernando Martins  sw.c               13/02/2012  Rotinas bases
*
 *  Fernando Martins  sw.h               13/05/2015 Sintonia para
protótipo         *
 *
\*********************************************************************
*******************/

#include "sw.h"

/*********************************************************************
*******************\
 *                       Definição de variáveis do software em memória
de programa             *
\*********************************************************************
*******************/



// - Globais ao sistema:

/*********************************************************************
*******************\
 *                            Definição de variáveis de flag
                  *
\*********************************************************************
*******************/
```

```
union unsigned_char flags_sw1;                          // Flags de
software
union unsigned_char flags_sw2;                          // Flags de
software


/*********************************************************************
******************\
 *                        Definição de variáveis do software em memória
de dados                *
\*********************************************************************
******************/

// - Globais ao software:

// - Globais ao sistema:

SM_BLDC sm_bldc = BLDC_PARALIZADO;
                                                        // 1+6,  2+6,
2+4,  3+4,  3+5,  1+5,   ( 0bABC )

/*********************************************************************
******************\
 *                                    Função principal
                    *
\*********************************************************************
******************/

int main( void )
{
    static unsigned int ponteiro_alvo = 0;
    static unsigned char contador = 0;

    #if( TIPO_SW == SW_TESTE )
        testa_firmware();
    #endif

    inicializa_sw();

    InitializeSystem();  // init usb

    #if defined(USB_INTERRUPT)
    USBDeviceAttach();
    #endif


    while( 1 )
    {
        ClrWdt();

        if( F_1MS )
        {
            F_1MS = 0;
            sm_processo_bldc();
        }

        if( F_10MS )
        {
            F_10MS = 0;
```

```
            le_fpga_poe_registros();

            contador++;
            if (contador > 1)
            {
                contador = 0;
                //dados.rpm_alvo = senoide_rpm[ponteiro_alvo++];
                if (ponteiro_alvo >= 300)
                {
                    ponteiro_alvo = 0;
                }
            }
        }

        if( F_PORTA_SERIAL_PCT_RX )
        {
            trata_pct_serial_rx();

            F_PORTA_SERIAL_PCT_RX = 0;
        }

    }

    return( 0 );
}


/*********************************************************************
******************\
 *                                  Implementação das funções
                          *
\*********************************************************************
******************/

void calcula_rpm( void  )
{
    unsigned char i = 0;
    signed long aux;
    static unsigned int buffer_rpm[30];
    static unsigned char ponteiro = 0;

    buffer_rpm[ponteiro++] = dados.contador_hall;
    dados.contador_hall = 0;
    if (ponteiro >= 30)
    {
        ponteiro = 0;
    }
    aux = 0;
    for ( i = 0 ; i < 30 ; i++)
    {
        aux += buffer_rpm[i];
    }

    if (dados.sentido_rotacao_real == SENTIDO_HORARIO)
    {
        dados.rpm_real = aux * 10;
    }
    else
    {
```

```
        dados.rpm_real = -aux * 10;
    }
}

/*********************************************************************
******************\
 * sm_processo_bldc
          *
 * Máquina de estado para controle do tempo e armazenamento
                  *
 *
*
 * Parâmetros: void
      *
 * Retorno   : void
      *
\*********************************************************************
******************/
inline void sm_processo_bldc( void )
{
    static unsigned int  esforco;
    static unsigned int  ponteiro;
    static unsigned int  timeout = TIMEOUT_BLDC;
    static unsigned int  timeout2 = TIMEOUT_PROTOCOLO_AUTOMATICO;
    static unsigned char buffer_final[ 17 ];// 1 caracter para sinal,
1 caracter para ponto decimal e outro para o fim da string
    unsigned char i;

    switch (sm_bldc)
    {
        case BLDC_PARALIZADO:
            ponteiro = 0;
            dados.pwm = 0;
            inicia_pid_controle( PID_SPEED );
            inicia_pid_controle( PID_CURRENT );
            // DEBUG finaliza_pid_controle( PID_SPEED );
            // DEBUG finaliza_pid_controle( PID_CURRENT );
            if  (dados.parametros.on == ON)
            {
                sm_bldc = BLDC_ON;
            }
            break;

        case BLDC_ON:

            timeout --;
            if (timeout == 0)
            {
                timeout = TIMEOUT_BLDC;

                calcula_rpm();
                calcula_rpm_medio();
                calcula_corrente_media();
                if ( dados.tipo_controle == PID_SPEED )
                {
                    if (dados.rpm_real_medio >= 0)
                    {
                        dados.sentido_rotacao = SENTIDO_HORARIO;
```

```
                        esforco = (algoritmo_pid_controle( PID_SPEED,
dados.rpm_alvo + 10000, dados.rpm_real + 10000 )); // delta de 20000
para aceitar rotações negativas
                    }
                    else
                    {
                        dados.sentido_rotacao = SENTIDO_ANTI_HORARIO;
                        esforco = (algoritmo_pid_controle(
PID_SPEED,dados.rpm_real + 10000, dados.rpm_alvo + 10000));
                    }
                }
                else
                {
                    if (dados.corrente_real_media >= 0)
                    {
                        dados.sentido_rotacao = SENTIDO_HORARIO;
                        esforco = (algoritmo_pid_controle(
PID_CURRENT, dados.corrente_alvo + 10000, dados.corrente_real_media +
10000 ));
                    }
                    else
                    {
                        dados.sentido_rotacao = SENTIDO_ANTI_HORARIO;
                        esforco = (algoritmo_pid_controle(
PID_CURRENT, dados.corrente_real_media + 10000, dados.corrente_alvo +
10000 ));
                    }
                }

                if ( esforco > 999)
                {
                    dados.pwm = 100;
                }
                else
                {
                    dados.pwm = esforco/10;
                }
            }

            if  (dados.parametros.on == OFF)
            {
                sm_bldc = BLDC_PARALIZADO;
            }

            timeout2 --;
            if (timeout2 == 0)
            {
                timeout2 = TIMEOUT_PROTOCOLO_AUTOMATICO;
                if ( dados.tipo_protocolo == SIMPLIFICADO)
                {
                    envia_resposta(ID_WR_SPEED_MEASURED,
dados.rpm_real_medio);
                    envia_resposta(ID_WR_CURRENT_MEASURED,
dados.corrente_real_media);
                }
            }
            break;
    }
}
```

```
void calcula_rpm_medio( void )
{
    static signed long acc[QTD_MEDIA_RPM];
    static unsigned char inicio = 1;
    static unsigned char ponteiro = 0;
    signed long aux;
    unsigned char i;

    if (inicio)
    {
        inicio = 0;
        ponteiro = 0;
        for (i = 0; i < QTD_MEDIA_RPM; i++)
        {
            acc[QTD_MEDIA_RPM] = 0;
        }
    }

    acc[ponteiro++] = dados.rpm_real;

    if (ponteiro >= QTD_MEDIA_RPM )
    {
        ponteiro = 0;
    }

    aux = 0;

    for (i = 0; i < QTD_MEDIA_RPM; i++)
    {
        aux += acc[i];
    }

    dados.rpm_real_medio = aux / QTD_MEDIA_RPM;

}

void calcula_corrente_media( void )
{
    static signed long acc[QTD_MEDIA_CORRENTE];
    static unsigned char inicio = 1;
    static unsigned char ponteiro = 0;
    signed long aux;
    unsigned char i;

    if (inicio)
    {
        inicio = 0;
        ponteiro = 0;
        for (i = 0; i < QTD_MEDIA_CORRENTE; i++)
        {
            acc[QTD_MEDIA_CORRENTE] = 0;
        }
    }

    acc[ponteiro++] = adc.ifase;

    if (ponteiro >= QTD_MEDIA_CORRENTE )
    {
```

```
        ponteiro = 0;
    }

    aux = 0;

    for (i = 0; i < QTD_MEDIA_CORRENTE; i++)
    {
        aux += acc[i];
    }



    if ((dados.sentido_rotacao_real == SENTIDO_ANTI_HORARIO) ||
(dados.freio == 1))
    {
        dados.corrente_real_media = -aux / (QTD_MEDIA_CORRENTE/8);
    }
    else
    {
        dados.corrente_real_media = aux / (QTD_MEDIA_CORRENTE/8);
    }
}


/*************************************************************************
******************\
 * inicializa_sw
*
 * Rotina de inicialização do software
*
 *
*
 * Parâmetros: void
*
 * Retorno   : void
*
\*************************************************************************
******************/
void inicializa_sw( void )
{
    unsigned char i;

    inicializa_fw();

    // Zera flags
    flags_sw1.value = 0;
    flags_sw2.value = 0;

    //inicializa_memoria_dados_eeprom();;

        // Inicializa módulo controle
    inicializa_controle();

    #if( defined( PID_SPEED ) )
        // Carrega parâmetros de controle da bomba de fluxo positivo
        controle.pid[ PID_SPEED ].N  = PID_SPEED_N;      // Pólo
limitador do ganho derivativo
        controle.pid[ PID_SPEED ].H  = PID_SPEED_H;      // Intervalo
de amostragem para PID, em segundos
```

```
        controle.pid[ PID_SPEED ].TT = PID_SPEED_TT;     // Tempo de
tracking para o anti-windup
        controle.pid[ PID_SPEED ].bp = PID_SPEED_KP;     // Banda
proporcional
        controle.pid[ PID_SPEED ].ti = PID_SPEED_KI;     // Tempo
integral
        controle.pid[ PID_SPEED ].td = PID_SPEED_KD;     // Tempo
derivativo
    #endif

    #if( defined( PID_CURRENT ) )
        // Carrega parâmetros de controle da bomba de fluxo positivo
        controle.pid[ PID_CURRENT ].N  = PID_CURRENT_N;      // Pólo
limitador do ganho derivativo
        controle.pid[ PID_CURRENT ].H  = PID_CURRENT_H;      //
Intervalo de amostragem para PID, em segundos
        controle.pid[ PID_CURRENT ].TT = PID_CURRENT_TT;     // Tempo
de tracking para o anti-windup
        controle.pid[ PID_CURRENT ].bp = PID_CURRENT_KP;     // Banda
proporcional
        controle.pid[ PID_CURRENT ].ti = PID_CURRENT_KI;     // Tempo
integral
        controle.pid[ PID_CURRENT ].td = PID_CURRENT_KD;     // Tempo
derivativo
    #endif

    dados.pwm = 0;
    dados.rpm_alvo = 0;
    dados.rpm_real = 0;
    dados.corrente_alvo = 0;
    dados.corrente_real_media = 0;

    dados.tipo_controle = PID_SPEED;
    dados.tipo_protocolo = SIMPLIFICADO;
    dados.parametros.on = ON;
    dados.sentido_rotacao = SENTIDO_HORARIO;
}




/*********************************************************************
******************\
 * time_out
                                    *
 * Rotina para contar e indicaqr quanto o tempo estourar
*
 *
*
 * Parâmetros: tempo inicial
*
 * Retorno   : 0 tempo estourado, >0  contagem em andamento , -1 já
acabou              *
\*********************************************************************
******************/
unsigned int time_out ( unsigned char inicia, unsigned int
tempo_inicial)
{
```

```
    static unsigned int tempo = 0;

    if (inicia)
    {
        tempo = tempo_inicial;
    }
    else
    {
        if (tempo)          // maior que 0?
        {
            tempo--;
            return(tempo);                    // se zero , acabou
        }
        else                // ja indicou estouro , agora é erro
        {
            return(-1);
        }
    }
return(2);   // em andamento
}
```

```
/*********************************************************************
*******************\
 *                          Software para controle de motor BLDC
*
 *
*
 *                      Desenvolvido por Fernando de Almeida Martins
*
 *
*
 *                           E-mail: fmartins_eng@yahoo.com
*
 *
*
 * Data: 15/04/2014
Versão: 1.0 *
 *
*
\*********************************************************************
*******************/

/*********************************************************************
*******************\
 *                                       Controle de Versões
*
 *
*
 * Responsável         Versão              Data      Descrição
*
 *
*
 * Fernando Martins  protocolo.h               15/04/2014  Rotinas
bases                          *
 *
*
 *
*
\*********************************************************************
*******************/

#ifndef PROTOCOLO_SERIAL_H
#define     PROTOCOLO_SERIAL_H

#include "..\fw\fw.h"                               // Arquivo de
definição de pinos, variáveis e funções do firmware

/*********************************************************************
*******************\
 *******
*********
 *******                                 MAPAS DE FLAGS
*********
 *******
*********
\*********************************************************************
*******************/

extern union unsigned_char flags_protocolo;                    //
Flags de software
```

```
/*****
 *    bit0:  F_PROTOCOLO_RESERVADO

 ****/

/*********************************************************************
******************\
 *                        Definição de variáveis do firmware em memória
de programa       *
\*********************************************************************
******************/

/*********************************************************************
******************\
 *                                    Definição de constantes
                        *
\*********************************************************************
******************/

#define HEADER                                  0x7F
#define NODE_ADDRESS                            0xF1

//Telemetry packet ID:
#define ID_WR_SPEED_MEASURED                    0x10     //RW speed
measured by shaft encoder (rpm)
#define ID_WR_CURRENT_MEASURED                  0x11     //RW motor
current measured (mA)
//#define ID_GYRO__ACCUMULATED_MEASURED       0x13   //Gyro
accumulated measurement
//#define ID_GYRO_RAW_MEASURED                  0x14   //Gyro raw
measurement
#define ID_STATUS                               0x15     //Status
//#define ID_TEMP                                 0x17   //Controller
temperature (°C)
//#define ID_VERSION                              0x18   //Software
version
//#define ID_WR_TEMPERATURE                       0x19   //RW temperature
(°C)
//#define ID_WR_PRESSURE                          0x1A   //RW pressure
(psi)
//#define ID_DELTA_ANGLE                          0x1B   //Inertial delta
angle telemetry (°)
//protocolo adicional INPE
#define ID_WR_SPEED_REFERENCE                    0x1C    //RW speed
measured REFERENCE (rpm)
#define ID_WR_CURRENT_REFERENCE                  0x1D    //RW motor
current REFERENCE (mA)

//Telecommand packet ID:
#define ID_RW_SPEED_CONTROL_ON                  0x40     //RW speed
control ON & speed reference (rpm)
#define ID_RW_CURRENTE_CONTROL_ON               0x41     //RW current
control ON & current reference (mA)
//#define ID_INERTIAL_RATE_CONTROL_ON           0x42   //Inertial rate
control ON & rate reference (°/s)
//#define ID_INERTIAL_ANGLE_CONTROL_ON          0x43   //Inertial ?
angle control ON & angle reference (°)
//#define ID_K1_GAIN_INERTIAL_RATE              0x44   //Gain K1 for
inertial rate PI controller
```

```
//#define ID_K2_GAIN_INERTIAL_RATE            0x45   //Gain K2 for
inertial rate PI controller
#define ID_K1_GAIN_SPEED                    0x47     //Gain K1 for
wheel speed PI controller
#define ID_K2_GAIN_SPEED                    0x48     //Gain K2 for
wheel speed PI controller
//#define ID_K1_GAIN_INERTIAL_ANGLE           0x49   //Gain K1 for
inertial delta angle P controller
#define ID_RW_GYRO_ON_OFF                   0x4A     //RW & Gyro
ON/OFF switch*
//#define ID_GYRO_OFFSET_BIAS                 0x4B   //Gyro offset
(bias) correction

/************************************************\
 *              Diretivas de compilação          *
\************************************************/

/************************************************\
 *                      Flags                     *
\************************************************/

#define F_PROTOCOLO_RESERVADO          flags_protocolo.bit0   //
Indica se o módulo I2C master por HW já foi inicializado (é
necessário, pois mais de um módulo pode chamar a rotina de
inicialização e isto pode causar problemas)

/************************************************\
 *                    constantes e estruturas                    *
\************************************************/


/*********************************************************************
******************\
 *                     Definição de variáveis do módulo em memória de
programa                *
\*********************************************************************
******************/

// - Globais ao sistema:


// - Globais ao módulo:


/*********************************************************************
******************\
 *                       Definição de variáveis do módulo em memória
de dados               *
\*********************************************************************
******************/

// - Globais ao sistema:


// - Globais ao módulo:


/*********************************************************************
******************\
```

```
 *                                                    Macros
                            *
\*********************************************************************
*******************/


/*********************************************************************
*******************\
 *                                                 Prototipagem
                                  *
\*********************************************************************
*******************/
inline void trata_pct_serial_rx( void );
void trata_protocolo_serial(void);
void envia_resposta_curta( unsigned char id );
void envia_resposta( unsigned char id, signed long data );

/*************************************************************\
 *                                Funções do módulo
          *
\*************************************************************/

#endif      /* PROTOCOLO_SERIAL_H */
```

```
/*********************************************************************
*******************\
 *                          Software para controle de motor BLDC
*
 *
*
 *                      Desenvolvido por Fernando de Almeida Martins
*
 *
*
 *                           E-mail: fmartins_eng@yahoo.com
*
 *
*
 * Data: 15/04/2014
Versão: 1.0 *
 *
*
\*********************************************************************
*******************/

/*********************************************************************
*******************\
 *                                       Controle de Versões
*
 *
*
 * Responsável         Versão              Data      Descrição
*
 *
*
 * Fernando Martins  protocolo.h               15/04/2014  Rotinas
bases                          *
 *
*
 *
*
\*********************************************************************
*******************/


#include "protocolo_serial.h"


/*********************************************************************
*******************\
 *                                            Flags do módulo
*
\*********************************************************************
*******************/

union unsigned_char flags_protocolo;


/*********************************************************************
*******************\
 *                       Definição de variáveis do módulo em memória de
programa               *
```

```
\*********************************************************************
*******************/

// - Globais ao módulo:


// - Globais ao sistema:



/*********************************************************************
*******************\
 *                          Definição de variáveis do módulo em memória
de dados                *
\*********************************************************************
*******************/

// - Globais ao módulo:



typedef union {
      long llong;
      float ffloat;
} Unionlonttofloat;



// - Globais ao sistema:



/*********************************************************************
*******************\
 *                                          Funções estáticas
      *
\*********************************************************************
*******************/

/*********************************************************************
*******************\
 *                                  Vetores de interrupção
*
\*********************************************************************
*******************/


/*********************************************************************
*******************\
 *                                     Implementação das funções
                           *
\*********************************************************************
*******************/

/*********************************************************************
*******************\
 *
*
```

```
 * Rotina para iniciar o protocolo
*
 *
*
 * Parâmetros: void
*
 * Retorno   : void
*
\*****************************************************************
******************/
// Data packet
//The data packet structure is used for
//Telemetry responses (Packet ID = 0x10 to 0x3F) that originate in the
slave node
//Telecommands (Packet ID = 0x40 to 0x6F) that originate in the master
node
//
//Telecommand: Command the x-axis reaction wheel to go to 1000 rpm.
//The master node, or desktop computer, transmits the wheel speed
telecommand
//(packet ID 0x40) to the slave node, or x-axis of the reaction wheel
and gyroscope electronics,
//(address 0xF1) with the telecommand data set to the 32-bit floating
point representation for a value of 1000:
//Header    Address     Packet ID   Data
//0x7F          0xF1            0x40            0x00 0x00  0x7A  0x44
//
//The slave node, or x-axis of the reaction wheel and gyroscope
electronics, (address 0xF1) responds with a wheel
//telecommand acknowledge (packet ID 0x40) within 20 milliseconds:
//Header    Node address      Packet ID
//0x7F          0xF1            0x40

//Short packet (no data)
//The short packet structure is used for
//Telemetry requests (Packet ID = 0x10 to 0x3F) that originate in the
master node
//Telecommand acknowledges (Packet ID = 0x40 to 0x6F) that originate
in the slave node
//
//Telemetry request: Request the wheel speed of the x-axis reaction
wheel
//The master node, or desktop computer, transmits the wheel speed
telemetry request (packet ID 0x10)
//to the slave node, or x-axis of the reaction wheel and gyroscope
electronics, (address 0xF1):
//Header    Address     Packet ID
//0x7F          0xF1            0x10
//
//The slave node, or x-axis of the reaction wheel and gyroscope
electronics, (address 0xF1)
//responds with a wheel speed telemetry response (packet ID 0x10) with
the telemetry data set to
//the 32-bit floating point representation of the current wheel speed,
i.e. 999 rpm, within 20 milliseconds:
//Header    Address         Packet ID     Data
//0x7F          0xF1            0x10            0x00 0xC0  0x79  0x44

//
```

```
//Telemetry packet ID:
//0x10      RW speed measured by shaft encoder (rpm)
//0x11      RW motor current measured (mA)
//0x12
//0x13      Gyro accumulated measurement
//0x14      Gyro raw measurement
//0x15      Status
//0x16
//0x17      Controller temperature (°C)
//0x18      Software version
//0x19      RW temperature (°C)
//0x1A      RW pressure (psi)
//0x1B      Inertial delta angle telemetry (°)
// protocolo adicional INPE
//0x1C      request speed reference (rpm)
//0x1D      request current reference (mA)

//
//Telecommand packet ID:
//0x40      RW speed control ON & speed reference (rpm)
//0x41      RW current control ON & current reference (mA)
//0x42      Inertial rate control ON & rate reference (°/s)
//0x43      Inertial ? angle control ON & angle reference (°)
//0x44      Gain K1 for inertial rate PI controller
//0x45      Gain K2 for inertial rate PI controller
//0x46
//0x47      Gain K1 for wheel speed PI controller
//0x48      Gain K2 for wheel speed PI controller
//0x49      Gain K1 for inertial delta angle P controller
//0x4A      RW & Gyro ON/OFF switch*
//0x4B      Gyro offset (bias) correction

//Reaction wheel and gyroscope on/off telecommand
//The individual power to the reaction wheel RWSS and the gyroscope
GSS can be switched on and off separately by a serial communications
telecommand from the master node (packet ID 0x4A).
//Telecommand: Switch both the reaction wheel RWSS and the gyroscope
GSS off.
//Header    Node address      Packet ID   Data
//0x7F          0xF1            0x4A            0    0     0     0
//
//Telecommand: Switch the reaction wheel RWSS on and the gyroscope GSS
off.
//Header    Node address      Packet ID   Data
//0x7F          0xF1            0x4A            1    0     0     0


//Current control mode
//When the master node transmits a current reference telecommand
(packet ID 0x41),
//the current control mode is selected and the slave node controls the
reaction
//wheel armature current to follow the reference current. The current
control mode
//is useful since a constant armature current is approximately
equivalent to a constant
//torque applied to the reaction wheel rotor.
//(However, the current control mode is not strictly equivalent to a
torque control mode,
```

```
//since the dynamic friction of the reaction wheel decreases the
effective wheel
//torque at higher wheel speeds.)
//The value of the current reference must be between ?2200 and +2200
mA. The master node may
//transmit current reference telecommands to the slave node at a
maximum rate of 10 Hz
//(sampling time = 100 ms).
//8.2 Speed control mode
//
//When the master node transmits a speed reference telecommand (packet
ID 0x40),
//the speed control mode is selected and the slave node controls the
reaction
//wheel speed to follow the reference speed. The speed control mode is
useful
//since the torque applied to the spacecraft body can be controlled
more accurately
//by commanding the wheel speed to track a reference wheel speed
profile. Accurate wheel
//speed control is also equivalent to accurate wheel momentum control.
//The wheel speed controller is a PI controller with a sampling time
of Ts = 100ms.
//The master node can set the wheel speed controller gains and by
transmitting their corresponding
//telecommands (packet ID 0x47 and 0x48 respectively). However, it
should not be necessary to change
//the wheel speed controller gains under normal circumstances, since
the optimal default gains have
//already been chosen for the specific reaction wheel. The default
gains for the wheel speed
//controller are and . k1=2000 e k2 = -1900
//The value of the speed reference must be between ?4200 and +4200
rpm. The master node may
//transmit speed reference telecommands to the slave node at a maximum
rate of 1 Hz
//(sampling time = 1 second).


/*
 MUDA SPEED 1000RPM
7F F1 40 00 00 7A 44

SPEED 2000 RPM
7F F1 40 00 00 FA 44

SPEED -1000RPM
7F F1 40 00 00 7A C4

CORRENTE 100MA
7F F1 41 00 00 C8 42

CORRENTE 200MA
7F F1 41 00 00 48 43

REQUEST SPEED
7F F1 10

REQUEST CURRENT
```

```
7F F1 11
*/
/*********************************************************************
******************\
 * trata_pct_serial_rx
*
 * Rotina para tratamento do pacote serial recebido. Os pacotes
recebidos aqui referem- *
 * se aos dados enviados pelo leitor de código de barras
*
 *
*
 * Parâmetros: void
*
 * Retorno   : void
*
\*********************************************************************
******************/
inline void trata_pct_serial_rx( void )
{
    if (( uart_1.rx[ 0 ] == HEADER) && ( uart_1.rx[ 1 ] ==
NODE_ADDRESS) )
    {
        trata_protocolo_serial();
        dados.tipo_protocolo = SUNSPACE; // a partir de então começo a
trabalhar no protocolo sunspace
    }
    uart_1.tam_rx = 0;
}

void trata_protocolo_serial(void)
{
    union unsigned_char aux;
    static unsigned char buffer_serial[5];
    Unionlonttofloat longtofloat;

    buffer_serial[0] = uart_1.rx[ 2 ];
    buffer_serial[1] = uart_1.rx[ 3 ];
    buffer_serial[2] = uart_1.rx[ 4 ];
    buffer_serial[3] = uart_1.rx[ 5 ];
    buffer_serial[4] = uart_1.rx[ 6 ];
    uart_1.tam_rx = 0;


    switch (buffer_serial[0])
    {
        case ID_WR_SPEED_MEASURED:                      //RW speed
measured by shaft encoder (rpm)
            envia_resposta(ID_WR_SPEED_MEASURED,
dados.rpm_real_medio);
            break;

        case ID_WR_CURRENT_MEASURED:                    //RW motor
current measured (mA)
            envia_resposta(ID_WR_CURRENT_MEASURED,
dados.corrente_real_media);
            break;

        case ID_STATUS:                                 //Status
```

```
            aux.value = 0;
            aux.bit1 = (dados.parametros.on? 1 : 0);            //
Bit 1:      RW On.
            aux.bit4 = (dados.tipo_controle? 0 : 1);            //
Bit 4:      Current loop active.
            aux.bit5 = (dados.tipo_controle? 1 : 0);            //
Bit 5:      Speed loop active.
            envia_resposta(ID_STATUS, aux.value  );
            break;

        case ID_RW_SPEED_CONTROL_ON:                 //RW speed
control ON & speed reference (rpm)
            longtofloat.llong = buffer_serial[1] +
buffer_serial[2]*0x100 + buffer_serial[3]*0x10000 +
buffer_serial[4]*0x1000000;
            dados.rpm_alvo = (unsigned long)longtofloat.ffloat;
            envia_resposta_curta(ID_RW_SPEED_CONTROL_ON);
            dados.tipo_controle = 0; //PID_SPEED;
            break;

        case ID_RW_CURRENTE_CONTROL_ON:                 //RW current
control ON & current reference (mA)
            longtofloat.llong = buffer_serial[1] +
buffer_serial[2]*0x100 + buffer_serial[3]*0x10000 +
buffer_serial[4]*0x1000000;
            dados.corrente_alvo = (unsigned long)longtofloat.ffloat;
            envia_resposta_curta(ID_RW_CURRENTE_CONTROL_ON);
            dados.tipo_controle = 1; //PID_CURRENT;
            break;

        case ID_K1_GAIN_SPEED:                          //Gain K1 for
wheel speed PI controller
            longtofloat.llong = buffer_serial[1] +
buffer_serial[2]*0x100 + buffer_serial[3]*0x10000 +
buffer_serial[4]*0x1000000;
            controle.pid[0].bp = (unsigned long)longtofloat.ffloat;
            envia_resposta_curta(ID_K1_GAIN_SPEED);
            break;

        case ID_K2_GAIN_SPEED:                          //Gain K2 for
wheel speed PI controller
            longtofloat.llong = buffer_serial[1] +
buffer_serial[2]*0x100 + buffer_serial[3]*0x10000 +
buffer_serial[4]*0x1000000;
            controle.pid[0].ti = longtofloat.ffloat;
            envia_resposta_curta(ID_K2_GAIN_SPEED);
            break;

        case ID_RW_GYRO_ON_OFF:                     //RW & Gyro
ON/OFF switch*
            dados.parametros.on = (buffer_serial[1]? 1 : 0);
            envia_resposta_curta(ID_RW_GYRO_ON_OFF);
            break;
        case ID_WR_SPEED_REFERENCE:
            envia_resposta(ID_WR_SPEED_REFERENCE, dados.rpm_alvo);
            break;
        case ID_WR_CURRENT_REFERENCE:
            envia_resposta(ID_WR_CURRENT_REFERENCE,
dados.corrente_alvo);
```

```
            break;
        default:
            break;

    }

}

void envia_resposta_curta( unsigned char id )
{
    porta_serial.tx[0] = HEADER;
    porta_serial.tx[1] = NODE_ADDRESS;
    porta_serial.tx[2] = id;
    porta_serial.tam_tx = 3;
    tx_pacote_porta_serial();

}

void envia_resposta( unsigned char id, signed long data )
{
    Unionlonttofloat longtofloat;
    static unsigned char tam_pacote_atual = 0;

    tam_pacote_atual = porta_serial.tam_tx;
    longtofloat.ffloat = data;

    porta_serial.tx[tam_pacote_atual] = HEADER;
    porta_serial.tx[tam_pacote_atual + 1] = NODE_ADDRESS;
    porta_serial.tx[tam_pacote_atual + 2] = id;
    porta_serial.tx[tam_pacote_atual + 3] = longtofloat.llong;
    porta_serial.tx[tam_pacote_atual + 4] = longtofloat.llong >> 8;
    porta_serial.tx[tam_pacote_atual + 5] = longtofloat.llong >> 16;
    porta_serial.tx[tam_pacote_atual + 6] = longtofloat.llong >> 24;

    porta_serial.tam_tx = tam_pacote_atual + 7;
    tx_pacote_porta_serial();
}

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

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

/***********************************************/
```

```
/*********************************************************************
*******************\
 *                                        Módulo Controle

*
\*********************************************************************
*******************/

#ifndef _CONTROLE_H_
#define _CONTROLE_H_

#define MOD_CONTROLE

#include "..\..\fw\fw.h"                              // Arquivo de
definição de pinos, variáveis e funções do firmware


/*********************************************************************
*******************\
 *******
*********
 *******                                MAPAS DE FLAGS
*********
 *******
*********
\*********************************************************************
*******************/

extern union unsigned_char flags_controle;          // Definição de
flags do
/*********************************************************************
*******************\
 *                                      Definição de constantes
                     *
\*********************************************************************
*******************/

/*******************************************\
 *                   Flags:                 *
\*******************************************/

/*******************************************\
 *                 Auxiliares:              *
\*******************************************/

// Número de controladores ONOFF
#define CONTROLE_TOTAL_ONOFF                0

// Número de controladores PID
#define CONTROLE_TOTAL_PID                  2

// Ação do controle ONOFF
#define ONOFF_ACAO_NORMAL                   0
#define ONOFF_ACAO_REVERSA                  1


/*********************************************************************
*******************\
```

```
 *                                  Definição de estruturas do módulo
                    *
\*********************************************************************
******************/

typedef struct
{
    struct
    {
        unsigned char on : 1;

        float N;
        float H;
        float TT;

        float bp;                               // x 10
        float ti;
        float td;
    } pid[ CONTROLE_TOTAL_PID ];

    struct
    {
        unsigned char on   : 1;
        unsigned char acao : 1;

        int histerese;                                  // x 10
    } onoff[ CONTROLE_TOTAL_ONOFF ];
} Controle;


/*********************************************************************
******************\
 *                      Definição de variáveis do módulo em memória de
programa              *
\*********************************************************************
******************/

// - Globais ao sistema:

/*********************************************************************
******************\
 *                        Definição de variáveis do módulo em memória
de dados              *
\*********************************************************************
******************/

// - Globais ao sistema:
extern Controle controle;


/*********************************************************************
******************\
 *                                        Macros
                                *
\*********************************************************************
******************/
```

```
/*********************************************************************
******************\
 *                                          Prototipagem
                                 *
\*********************************************************************
******************/

inline void inicializa_controle( void );

void inicia_pid_controle( unsigned char n );
inline void finaliza_pid_controle( unsigned char n );
unsigned int algoritmo_pid_controle( unsigned char n, float sp, float
y );

void inicia_onoff_controle( unsigned char n );
inline void finaliza_onoff_controle( unsigned char n );
unsigned int algoritmo_onoff_controle( unsigned char n, float sp,
float pv );


#endif // _CONTROLE_H_
```

```
/*****************************************************************
******************\
 *                                        Módulo Controle
*
*
*
\*****************************************************************
******************/

#include "mod_controle.h"                          // Arquivo de
definição variáveis e funções do módulo Controle


/*****************************************************************
******************\
 *                                         Flags do módulo
*
\*****************************************************************
******************/

union unsigned_char flags_controle;


/*****************************************************************
******************\
 *                      Definição de variáveis do módulo em memória de
programa              *
\*****************************************************************
******************/

// - Globais ao módulo:


// - Globais ao sistema:



/*****************************************************************
******************\
 *                        Definição de variáveis do módulo em memória
de dados              *
\*****************************************************************
******************/

// - Globais ao módulo:
struct
{
    unsigned char estado : 1;                  // Estado: ON/OFF
    unsigned char inicio : 1;                  // Início do controle

    unsigned int saida;
} onoff[ CONTROLE_TOTAL_ONOFF ];

struct
{
    unsigned char estado : 1;                  // Estado: ON/OFF
    unsigned char inicio : 1;                  // Início do controle

    float ad;                                  // Termo ad
```

```
    float bd;                                       // Termo bd
    float bi;                                       // Termo bi
    float bt;                                       // Termo bt
    float d_ant;                                    // d(n-1)
    float y_ant;                                    // pv(n-1)
    float u;                                        // Esforço de controle
com saturação (0..100%)
    float v;                                        // Esforço de controle
sem saturação
    float p_n;                                      // Termo proporcional
    float i_n;                                      // Termo integral
    float d_n;                                      // Termo derivativo
} pid[ CONTROLE_TOTAL_PID ];

// - Globais ao sistema:
Controle controle;


/*****************************************************************
******************\
 *                                      Funções estáticas
     *
\*****************************************************************
******************/




/*****************************************************************
******************\
 *                                Vetores de interrupção
*
\*****************************************************************
******************/




/*****************************************************************
******************\
 *                                  Implementação das funções
                          *
\*****************************************************************
******************/

/*****************************************************************
******************\
 * inicializa_controle
*
 * Rotina para iniciar o módulo Controle
*
 *
*
 * Parâmetros: void
*
 * Retorno   : void
*
\*****************************************************************
******************/
```

```
void inicializa_controle( void )
{
    unsigned char i;

    flags_controle.value = 0;                       // Zera flags do
módulo Controle

    // Zera variáveis
    for( i = 0 ; i < CONTROLE_TOTAL_ONOFF ; i++ )
    {
        onoff[ i ].estado = OFF;
        onoff[ i ].inicio = 1;

        onoff[ i ].saida = 0;

        controle.onoff[ i ].on = 0;
        controle.onoff[ i ].acao = ONOFF_ACAO_NORMAL;
    }

    for( i = 0 ; i < CONTROLE_TOTAL_PID ; i++ )
    {
        pid[ i ].estado = OFF;
        pid[ i ].inicio = 1;

        controle.pid[ i ].N  =  1.0;
        controle.pid[ i ].TT = 20.0;
        controle.pid[ i ].H  =  0.2;

        controle.pid[ i ].on = 0;
    }
}


/*********************************************************************
**********\
 * inicia_pid_controle
*
 * Rotina de liberação do algoritmo de PID
*
 *
*
 * Parâmetros: índice do controlador PID
*
 * Retorno:    void
*
\*********************************************************************
**********/
void inicia_pid_controle( unsigned char n )
{
    controle.pid[ n ].on = 1;

    pid[ n ].estado = ON;
    pid[ n ].inicio = 1;

    pid[ n ].i_n   = 0.0;
    pid[ n ].d_n   = 0.0;
    pid[ n ].d_ant = 0.0;
    pid[ n ].y_ant = 0.0;
}
```

```
/*********************************************************************
***********\
 * finaliza_pid_controle
*
 * Rotina de finalização do algoritmo de PID
*
 *
*
 * Parâmetros: índice do controlador PID
*
 * Retorno:    void
*
\*********************************************************************
***********/
inline void finaliza_pid_controle( unsigned char n )
{
    controle.pid[ n ].on = 0;

    pid[ n ].estado = OFF;
}


/*********************************************************************
***********\
 * algoritmo_pid_controle
*
 * Rotina de execução do algoritmo de controle PID
*
 *
*
 * Parâmetros: índice do controlador PID, set-point e variável de
controle (em  *
 *             ponto flutuante)
*
 * Retorno:    esforço do controle, de 0 a 1000
*
\*********************************************************************
***********/
unsigned int algoritmo_pid_controle( unsigned char n, float sp, float
y )
{
    if( pid[ n ].estado == ON )
    {
        // Cálculo do termo ad
        pid[ n ].ad = controle.pid[ n ].td / ( controle.pid[ n ].td +
( controle.pid[ n ].N * controle.pid[ n ].H ) );

        // Cálculo do termo bd
        pid[ n ].bd = ( 100.0 /  controle.pid[ n ].bp ) *
controle.pid[ n ].N * pid[ n ].ad;

        // Se Ti = 0, garante ação integral nula
        if( controle.pid[ n ].ti == 0 )
        {
            pid[ n ].bi  = 0.0;
            pid[ n ].i_n = 0.0;
            pid[ n ].bt  = 0.0;
```

```
        }
        else
        {
            pid[ n ].bi = ( ( 100.0 /  controle.pid[ n ].bp  ) *
controle.pid[ n ].H ) / controle.pid[ n ].ti;
            pid[ n ].bt = controle.pid[ n ].H / controle.pid[ n ].TT;
        }

        // Cálculo do termo proporcional
        pid[ n ].p_n = ( 100.0 /  controle.pid[ n ].bp  ) * ( sp - y
);

        // Cálculo do termo derivativo (apenas se ainda não foi
calculado)
        if( !pid[ n ].inicio )
        {
            pid[ n ].d_n = ( pid[ n ].ad * pid[ n ].d_ant ) + ( pid[ n
].bd * ( pid[ n ].y_ant - y ) );

            // Atualiza regressor do termo derivativo
            pid[ n ].d_ant = pid[ n ].d_n;
        }
        else
        {
            pid[ n ].inicio = 0;
        }

        // Esforço de controle sem saturação
        pid[ n ].v = pid[ n ].p_n + pid[ n ].i_n + pid[ n ].d_n;

        // Esforço de controle com saturação
        if( pid[ n ].v < 0.0 )
        {
            pid[ n ].u = 0.0;
        }
        else if( pid[ n ].v > 100.0 )
        {
            pid[ n ].u = 100.0;
        }
        else
        {
            pid[ n ].u = pid[ n ].v;
        }

        // i(n+1)
        pid[ n ].i_n = ( pid[ n ].bi * ( sp - y ) ) + ( pid[ n ].bt *
( pid[ n ].u - pid[ n ].v ) ) + pid[ n ].i_n;

        // Atualiza regressor da variável de processo
        pid[ n ].y_ant = y;

        // Retorna esforço do controle (0 a 100.0%)
        return( pid[ n ].u * 10.0 );
    }

    return( 0 );
}
```

```
/*********************************************************************
***********\
 * inicia_onoff_controle
*
 * Rotina de liberação do algoritmo de ONOFF
*
 *
*
 * Parâmetros: índice do controlador ONOFF
*
 * Retorno:    void
*
\*********************************************************************
***********/
void inicia_onoff_controle( unsigned char n )
{
    controle.onoff[ n ].on = 1;

    onoff[ n ].estado = ON;
    onoff[ n ].inicio = 1;

    onoff[ n ].saida  = 0;
}


/*********************************************************************
***********\
 * finaliza_onoff_controle
*
 * Rotina de finalização do algoritmo de ONOFF
*
 *
*
 * Parâmetros: índice do controlador ONOFF
*
 * Retorno:    void
*
\*********************************************************************
***********/
inline void finaliza_onoff_controle( unsigned char n )
{
    controle.onoff[ n ].on = 0;

    onoff[ n ].estado = OFF;
}


/*********************************************************************
***********\
 * algoritmo_onoff_controle
*
 * Rotina de execução do algoritmo de controle ONOFF
*
 *
*
 * Parâmetros: índice do controlador ONOFF, set-point e variável de
controle (em*
```

```
 *             ponto flutuante)
*
 * Retorno:    esforço do controle, de 0 a 1000
*
\*****************************************************************
***********/
unsigned int algoritmo_onoff_controle( unsigned char n, float sp,
float pv )
{
    int histerese;

    if( onoff[ n ].estado == ON )
    {
        if( onoff[ n ].inicio )
        {
            onoff[ n ].inicio = 0;
            onoff[ n ].saida = 0;
        }

        histerese = controle.onoff[ n ].histerese / 10;

        if( controle.onoff[ n ].acao == ONOFF_ACAO_REVERSA )
        {
            if( pv >= sp )
            {
                onoff[ n ].saida = 0;
            }
            else
            {
                if( pv <= ( sp - histerese ) )
                {
                    onoff[ n ].saida = 1000;
                }
            }
        }
        else
        {
            if( pv <= sp )
            {
                onoff[ n ].saida = 0;
            }
            else
            {
                if( pv >= ( sp + histerese ) )
                {
                    onoff[ n ].saida = 1000;
                }
            }
        }

        return( onoff[ n ].saida );
    }

    return( 0 );
}
```

```
/*********************************************************************
*******************\
 *                                    Módulo Capture
*
 *
*
 *
*
\*********************************************************************
*******************/

#include "mod_capture.h"                          // Arquivo de
definição de variáveis e funções do módulo


/*********************************************************************
*******************\
 *                                          Flags do módulo
*
\*********************************************************************
*******************/

union unsigned_char flags_capture;          // Flags do módulo Capture


/*********************************************************************
*******************\
 *                      Definição de variáveis do módulo em memória de
programa              *
\*********************************************************************
*******************/

// - Globais ao módulo:


// - Globais ao sistema:


/*********************************************************************
*******************\
 *                        Definição de variáveis do módulo em memória
de dados             *
\*********************************************************************
*******************/

// - Globais ao módulo:


// - Globais ao sistema:



/*********************************************************************
*******************\
 *                                    Funções estáticas
      *
\*********************************************************************
*******************/
```

```
/*********************************************************************
******************\
 *                                  Vetores de interrupção
*
\*********************************************************************
******************/
void __ISR(_EXTERNAL_1_VECTOR, IPL7) INT1Interrupt( void )
{
    if (INTCONbits.INT1EP == 1)
    {
        executa_sensoreamento_hall( 1 );
    }
    else
    {
        executa_sensoreamento_hall( 0 );
    }

    INTCONbits.INT1EP = ~INTCONbits.INT1EP;
    IFS0bits.INT1IF = 0;                                    // Limpa
flag de interrupção Input Capture 1
}

void __ISR(_EXTERNAL_2_VECTOR, IPL7) INT2Interrupt( void )
{
    dados.ultimo_hall = 2;
    executa_sensoreamento_hall( 0 );
    INTCONbits.INT2EP = ~INTCONbits.INT2EP;
    IFS0bits.INT2IF = 0;                                    // Limpa
flag de interrupção Input Capture 2
}

void __ISR(_EXTERNAL_3_VECTOR, IPL7) INT3Interrupt( void )
{
    dados.ultimo_hall = 3;
    executa_sensoreamento_hall( 0 );
    INTCONbits.INT3EP = ~INTCONbits.INT3EP;
    IFS0bits.INT3IF = 0;                                    // Limpa
flag de interrupção Input Capture 3
 }

/*********************************************************************
******************\
 *                                  Implementação das funções
                           *
\*********************************************************************
******************/

/*********************************************************************
******************\
 * inicializa_captures
*
 * Rotina de inicialização do módulo Input Capture
*
 *
*
 * Parâmetros: void
*
```

```
 * Retorno:    void
*
\*********************************************************************
******************/
void inicializa_captures( void )
{

    INTCON = 0x000E;
    IPC1bits.INT1IP = 7;
    IPC2bits.INT2IP = 7;
    IPC3bits.INT3IP = 7;
    IFS0bits.INT1IF = 0;
    IFS0bits.INT2IF = 0;
    IFS0bits.INT3IF = 0;
    IEC0bits.INT1IE = 1;
    IEC0bits.INT2IE = 1;
    IEC0bits.INT3IE = 1;

 }
```

```
/*********************************************************************
*******************\
 *                                          Módulo Timer
*
 *
*
*
*
 *
*
\*********************************************************************
*******************/

#include "mod_timer.h"                                    // Arquivo de
definição de variáveis e funções do módulo Timer

/*********************************************************************
*******************\
 *                                          Flags do módulo
*
\*********************************************************************
*******************/

volatile union unsigned_char flags_timer;

/*********************************************************************
*******************\
 *                      Definição de variáveis do módulo em memória de
programa              *
\*********************************************************************
*******************/

// - Globais ao módulo:


// - Globais ao sistema:



/*********************************************************************
*******************\
 *                        Definição de variáveis do módulo em memória
de dados              *
\*********************************************************************
*******************/

// - Globais ao módulo:
unsigned char tempo_10ms;                          // Contador para
base de tempo de 10ms
unsigned char tempo_50ms;                          // Contador para
base de tempo de 50ms
unsigned char tempo_100ms;                         // Contador para
base de tempo de 100ms
unsigned char tempo_500ms;                         // Contador para
base de tempo de 500ms
unsigned char tempo_1000ms;                        // Contador para
base de tempo de 1000ms

// - Globais ao sistema:
```

```
/*********************************************************************
******************\
 *                          Timer2 Interrupt Service Routine - 1ms
*
 *
*
 * Descrição: Base de tempo de 1ms
*
 * Prioridade: 2 (baixa)
*
\*********************************************************************
******************/
void __ISR(_TIMER_2_VECTOR, IPL4) _T2Interrupt( void )
{
    calcula_base_tempo();                              // Gera as bases
de tempo

    IFS0bits.T2IF = 0;                                 // Limpa flag de
interrupção Timer2
}

/*********************************************************************
******************\
 *                          Timer2 Interrupt Service Routine - 1mS
*
 *
*
 * Descrição: Base de tempo de 1mS
*
 * Prioridade: 7 ( máxima)
*
\*********************************************************************
******************/
void __ISR(_TIMER_3_VECTOR, IPL6) _T3Interrupt( void )
{
    IFS0bits.T3IF = 0;                                 // Limpa flag de
interrupção Timer2

    atualiza_pwm();                               // Gera as bases de
tempo
}


/*********************************************************************
******************\
 *                          Timer2 Interrupt Service Routine - 1mS
*
 *
*
 * Descrição: Base de tempo de 1mS
*
 * Prioridade: 7 ( máxima)
*
\*********************************************************************
******************/
void __ISR(_TIMER_4_VECTOR, IPL4) _T4Interrupt( void )
{
    static unsigned long contador_anterior;
```

```
    LED2 = ~LED2;

    IFS0bits.T4IF = 0;                                  // Limpa flag de
interrupção Timer4

    if (contador_anterior == dados.contador_hall)
    {
        if ((dados.rpm_alvo != 0) || (dados.corrente_alvo != 0))
        {
            atualiza_rpm ( 0 );
        }
    }
    contador_anterior = dados.contador_hall;
}


/*********************************************************************
******************\
 *                                  Implementação das funções
                          *
\*********************************************************************
******************/

/*********************************************************************
******************\
 * inicializa_timer
*
 * Rotina de inicialização do Timer2
*
 *
*
 * Parâmetros: void
*
 * Retorno   : void
*
\*********************************************************************
******************/
inline void inicializa_timer( void )
{
    flags_timer.value = 0;                              // Zera flags do
módulo Timer

    tempo_10ms    = T_10MS;                              // Inicializa
base de tempo de 10ms
    tempo_50ms    = T_50MS;                              // Inicializa
base de tempo de 50ms
    tempo_100ms   = T_100MS;                             // Inicializa
base de tempo de 100ms
    tempo_500ms   = T_500MS;                             // Inicializa
base de tempo de 500ms
    tempo_1000ms  = T_1000MS;                            // Inicializa
base de tempo de 1000ms

    /*** Timer2 - 1ms: tarefas gerais do firmware ***/

    T2CONSET = 0x00002030;                              // Timer2 parado,
prescale 1:8, 16bits, fonte de clock interno
    TMR2CLR = 0xFFFFFFFF;                               // Zera contador
```

```
    PR2 = VALOR_TMR2;                                   // Carrega período
do Timer2
    IPC2bits.T2IP = 4;                                  // Nível de
prioridade: 6 (quase máxima)
    IFS0bits.T2IF = 0;                                  // Limpa flag de
interrupção do Timer2
    IEC0bits.T2IE = 1;                                 // Habilita
interrupção Timer2
    T2CONbits.TON = 1;                                  // Liga Timer2

    /*** Timer3 - 10ms: Atualização para A/D ***/
    T3CONSET = 0x00002030;                              // Timer3 parado,
prescale 1:8, 16bits, fonte de clock interno
    TMR3CLR = 0xFFFFFFFF;                               // Zera contador
    PR3 = VALOR_TMR3;                                   // Carrega período
do Timer3
    IPC3bits.T3IP = 6;                                  // Nível de
prioridade: 6 (quase máxima)
    IFS0bits.T3IF = 0;                                  // Limpa flag de
interrupção do Timer3
    IEC0bits.T3IE = 1;                                 // Habilita
interrupção Timer2
    T3CONbits.TON = 1;                                  // Liga Timer3

    /*** Timer4 - 5uS: tarefa ATUALIZA RPM ***/      // 200kHz

    T4CON = 0x2070;                                     // Timer parado,
prescale 1:256, 16bits, fonte de clock interno
    TMR4  = 0x0000;                                     // Zera contador
    PR4   = VALOR_TMR4;                                 // Carrega período
do Timer2
    IPC4bits.T4IP = 4;                                  // Nível de
prioridade: 6 (quase máxima)
    IFS0bits.T4IF = 0;                                  // Limpa flag de
interrupção do Timer2
    IEC0bits.T4IE = 1;                                    // Habilita
interrupção Timer2
    T4CONbits.TON = 1;                                  // Liga Timer2
}


/*********************************************************************
******************\
 * calcula_base_tempo
*
 * Rotina para cálculo das bases de tempo do sistema
*
 *
*
 * Parâmetros: void
*
 * Retorno   : void
*
\*********************************************************************
******************/
void calcula_base_tempo( void )
{
    F_1MS = 1;                                          // Informa que
passou-se 1ms
```

```
    trata_uart();                                   // Envio de
pacotes de dados pela serial e teste de timeouts de transmissão e
recepção

    tempo_10ms--;                                   // Decrementa
tempo da base de 10ms
    if( !tempo_10ms )                               // Fim do tempo?
Sim
    {
        F_10MS = 1;                                 // Informa que
passou-se 10ms
        tempo_10ms = T_10MS;                        // Recarrega tempo

        tempo_50ms--;                               // Decrementa
tempo da base de 50ms
        if( !tempo_50ms )                           // Fim do tempo?
Sim
        {
            F_50MS = 1;                             // Informa que
passou-se 50ms
            tempo_50ms = T_50MS;                    // Recarrega tempo

            tempo_100ms--;                          // Decrementa
tempo da base de 100ms
            if( !tempo_100ms )                      // Fim do tempo?
Sim
            {
                F_100MS = 1;                        // Informa que
passou-se 100ms
                tempo_100ms = T_100MS;              // Recarrega tempo

                tempo_500ms--;                      // Decrementa
tempo da base de 500ms
                if( !tempo_500ms )                  // Fim do tempo?
Sim
                {
                    F_500MS = 1;                    // Informa que
passou-se 500ms
                    tempo_500ms = T_500MS;          // Recarrega tempo

                    tempo_1000ms--;                 // Decrementa
tempo da base de 1000ms
                    if( !tempo_1000ms )             // Fim do tempo?
Sim
                    {
                        F_1000MS = 1;               // Informa que
passou-se 1000ms
                        tempo_1000ms = T_1000MS;    // Recarrega tempo
                    }
                }
            }
        }
    }
}
```