UNIVERSIDADE FEEVALE

LEANDRO BORGES

PROTÓTIPO DE CONTROLADOR PARA MOTOR SEM ESCOVAS COM IMÃS PERMANENTES

Novo Hamburgo
2017

LEANDRO BORGES

PROTÓTIPO DE CONTROLADOR PARA MOTOR SEM ESCOVAS COM IMÃS PERMANENTES

Trabalho de Conclusão de Curso II apresentado como requisito parcial à obtenção de grau de Bacharel em Engenharia Eletrônica pela Universidade Feevale.

Orientador: DELFIM LUIZ TOROK

Novo Hamburgo
2017

Dedico este trabalho aos meus pais e amigos que sempre me incentivaram, e em especial a minha amada esposa, que esteve ao meu lado intensamente durante esta jornada .

AGRADECIMENTOS

Primeiramente, a minha amada esposa Morgana Boff. Fonte de inspiração dessa jornada. Sem a qual, o caminho seria mais árduo.

Ao meu pai Roque, minha mãe Clair, minha irmã Viviane e meus avos, que sempre me incentivaram a estudar, em busca de um futuro digno.

A minha cunhada Débora Vargas, pelo apoio incondicional ao longo dos anos.

A minha amiga de longa data Cíntia Nicodem e sua família, pelo apoio em muitos momentos.

A Família Tamanchievies, pelo acolhimento, incentivo e compreensão ao longo dos anos.

Aos meus professores, pois sem eles nada seria possível. Foram verdadeiros parceiros de jornada; em especial ao meu orientador, que dedicou várias horas em vários momentos, com muita atenciosidade e dedicação!

Aos meus colegas, por me ajudarem a ter forças a continuar a perseguir meus objetivos, nos desafiando e nos encorajando a buscar resultados cada vez mais complexos a medida que o curso avançava.

A Universidade Feevale, por proporcionar um ambiente tão favorável ao desenvolvimento de ideias.

"Que os vossos esforços desafiem as impossibilidades, lembrai-vos de que as grandes coisas do homem foram conquistadas do que parecia impossível..." (Charles Chaplin)

RESUMO

O presente Trabalho consiste no desenvolvimento de um protótipo de controlador para motor sem escovas com imãs permanentes, utilizando o microcontrolador dsPIC30F2010. Os motores sem escovas com imãs permanentes avançam no mercado mundial em diversos segmentos. Abre-se um nicho que demanda conhecimentos específicos de eletrônica na elaboração de seus controladores. O protótipo desenvolvido busca a aplicação da teoria de controle de motores sem escovas com imãs permanentes, aplicando topologias de controle temporal, circuito aberto e circuito fechado. Analisa-se as problemáticas de implantação e apresenta-se soluções baseadas nos fundamentos da eletrônica. Destaca-se, ao longo deste trabalho, as problemáticas devido as interferências elétricas em diversos componentes do sistema; a metodologia dos testes desenvolvidos buscando as causas; propõem uma solução e apresenta os resultados da aplicação dessa. O protótipo se demonstra viável, atende os princípios de funcionamento nas topologias aplicadas, apresenta-se como um promissor projeto acadêmico, dando base para futuros aprimoramentos e implementações de novas funções, sendo possível o desenvolvimento de um produto versátil para o mercado brasileiro e internacional.

Palavras-chave: Brushless. Microcontrolador. Dspic. Driver.

ABSTRACT

The present Work consists in the development of a prototype controller for brushless motor with permanent magnets using the dsPIC30F2010 microcontroller. Brushless motors with permanent magnets advance the world market in several segments. It opens a niche that demands specific knowledge of electronics in the elaboration of its controllers. The prototype developed seeks the application of the control theory of brushless motors with permanent magnets, applying topologies of temporal control, open-loop and closed-loop. Analyzes the implementation problems and present solutions based on the fundamentals of electronics. It is highlighted, throughout this work, the problems due to the electrical interference in several components of the system; the methodology of the tests developed seeking the causes; propose a solution and present the results of applying it. The prototype proves to be viable, meets the principles of operation in applied topologies, presents itself as a promising academic project, providing the basis for future improvements and implementations of new functions, being potential the development of a versatile product for the Brazilian and international market.

Keywords: Brushless. microcontrollers. Dspic. Driver.

LISTA DE ILUSTRAÇÕES

Figura 1 - Diagrama dos principais motores encontrados 12
Figura 2 - Estator de um motor BLDC 16
Figura 3 - Tipos de arranjos de rotores com imãs permanentes 17
Figura 4 - Fluxograma da rotina de sensoriamento em Dspic 18
Figura 5 - Sinais de sensor hall, corrente de fase, EMF e torque de saída 19
Figura 6 - Sequência de energização dos enrolamentos 20
Figura 7 - Bloco diagrama controlador de motor BLDC. 23
Figura 8 - Representação Motor BLDC simplificado. 23
Figura 9 - Representação da leitura do sensor contra o tempo de comutação 24
Figura 10 - Diagrama de circuito de saída com 3 fases 26
Figura 11 - dsPIC30F2010 com encapsulamento SMD 27
Figura 12 - Diagrama de pinos do dsPIC30f2010 encapsulamento SDIP 27
Figura 13 - Diagrama de blocos internos do dsPIC30F2010 28
Figura 14 - Diagrama de comutação BLDC 30
Figura 15 - Bloco diagrama acionamento motor BLDC 32
Figura 16 - PWM alinhado 34
Figura 17 - Exemplo de saida PWM em relação ao OVDCON 35
Figura 18 - Diagrama de bloco do microcontrolador 36
Figura 19 - Ambiente de desenvolvimento MPLAB IDE 37
Figura 20 - Programador Pickit 3 38
Figura 21 - Fluxograma de inicialização do controle de malha aberta 40
Figura 22 - Esquemático controle de malha aberta 40
Figura 23 - Fluxograma loop principal 41
Figura 24 - Diagrama de bloco de malha fechada 42
Figura 25 - Fluxograma da inicialização de programa em malha fechada 42
Figura 26 - Sinais de Modulação por Largura de Pulso 43
Figura 27 - Diagrama de blocos do hardware 45
Figura 28 - Motor no.: CO-BL23B46-02-02RO 46
Figura 29 - Esquemático placa de controle 48
Figura 30 - Vista superior Placa de Controle na matriz de contatos 49
Figura 31 - Barramento Placa de controle 49
Figura 32 - Esquemático módulo de potência 52
Figura 33 - Esquemático regulador LM7815 53
Figura 34 - Diagrama Funcionamento IR2101 54
Figura 35 - Esquemático high side e low side drive 54
Figura 36 - Esquemático acelerador com potenciômetro 55

Figura 37 - Diagrama Funcionamento ACS712 56
Figura 38 - Esquemático pull-up para o sensor hall 57
Figura 39 - Esquemático detector de tensão 58
Figura 40 - Vista superior em 3D do layout com placas sobrepostas 59
Figura 41 - Esquemático módulo de potência modificado 60
Figura 42 - Circuito montado na matriz de contatos 61
Figura 43 - Esquemático módulo de potência auxiliar 62
Figura 44 - Vista superior em 3D do layout da placa auxiliar 62
Figura 45 - Placa auxiliar conectada ao motor 63
Figura 46 - Diagrama de roteamento dos componentes placa de potência 64
Figura 47 - Vista superior em 3D do layout 65
Figura 48 - Placa de potência 66
Figura 49 - Vista inferior placa driver de potência 67
Figura 50 - Disposição e acoplamento do hardware 68
Figura 51 - Código fonte da rotina principal 69
Figura 52 - Fluxograma do motor em funcionamento 71
Figura 53 - Código inicialização do ADC 72
Figura 54 - Trecho de código com vetores de sentido da rotação 72
Figura 55 - Trecho de código para iniciar o motor 73
Figura 56 - Trecho de código de parada do motor 73
Figura 57 - Interferência durante rotação do motor 75
Figura 58 - Comparação de sinais após aplicação do filtro 76
Figura 59 - Análise dos sinal de fase do motor e sensor hall com interferência 77
Figura 60 - Blindagem dos condutores do sensor hall 78
Figura 61 - Interferência na carcaça do motor em funcionamento 78
Figura 62 - Condutor especial para aterramento da carcaça do motor 79
Figura 63 - Análise dos sinais do sensor hall sem interferência 80
Figura 64 - Largura de pulso sem carga no eixo e velocidade fixa 81
Figura 65 - Largura de pulso com carga no eixo 82
Figura 66 - Bancada de teste de validação em malha fechada 83

LISTA DE TABELAS

Tabela 1 - Comparando motor brushless com um motor CC com escovas 14
Tabela 2 - Sequência de rotação, sentido anti-horário 21
Tabela 3 - Sequência de rotação, sentido horário 21
Tabela 4 - Exemplo de tabela OVDCON 35
Tabela 5 - Especificações motor brushless 46
Tabela 6 - Diagrama do conector do motor sem escovas 47
Tabela 7 - Sequência chaveamento Motor no.: CO-BL23B46-02-02RO 47
Tabela 8 - Mapeamento das ligações do dsPIC30F2010 51
Tabela 9 - Tabela de identificação de condição LED. 70
Tabela 10 - Comparativo de carga com 900 RPM 83
Tabela 11 - Comparativo de malha fechada com 1200 RPM 84

LISTA DE ABREVIATURAS E SIGLAS

A	Amperes
ADC	Analog to Digital Converter
BEMF	Back Electromagnetic field
BLDC	Brushless Direct Current
CC	Corrente contínua
CPU	Central Processing Unit
DSP	Digital Signal Processor
EEPROM	Electrically Erasable Programmable Read Only Memory
EMF	Electromagnetic field
GND	Ground
IGBT	Isolated Gate Bipolar Transistor
MC	Motor Control
MIPS	Milhões de instruções por segundo
MMF	Magnetomotive Force
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
PI	Proportional integral
PIC	Programmable Interface Controller
PID	Proportional integral derivative
PPR	Pulse Per Revolution
PWM	Pulse Width Modulation
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
SMD	Surface Mount Device
TCC	Trabalho de Conclusão de Curso
TTL	Transistor-Transistor Logic

USB	Universal Serial Bus
V	Volts
VDD	Voltage Drain Drain
VSS	Voltage Source Source

SUMÁRIO

1 **INTRODUÇÃO** 11
1.1 OBJETIVO GERAL 11
1.2 OBJETIVOS ESPECÍFICOS 11
2 **FUNDAMENTAÇÃO TEÓRICA** 12
2.1 MOTOR ELÉTRICO 12
2.1.1 **MOTOR DE CORRENTE CONTÍNUA SEM ESCOVAS** 12
2.1.1.1 Construção e princípio de funcionamento 15
2.1.1.2 Estator 15
2.1.1.3 Rotor 16
2.1.1.4 Sensor de posicionamento 17
2.1.1.4.1 *Sensor hall* 17
2.1.1.5 Teoria de operação 18
2.1.1.5.1 *Sequência de Comutação* 19
2.1.1.5.2 *Controle de malha fechada* 22
2.2 MICROCONTROLADOR 26
2.2.1 **DSPIC** 26
2.2.1.1 dsPIC30F2010 27
2.2.1.1.1 *Técnicas sem sensor para a comutação do motor brushless* 29
2.2.1.1.2 *Técnicas com sensor para a comutação do motor brushless* 31
2.2.1.1.3 *Descrição de hardware para driver específico* 35
2.2.1.2 MPLAB IDE 37
2.2.1.3 Pickit 3 38
2.3 LINGUAGEM C 38
2.3.1 **DESCRIÇÃO FIRMWARE** 39
2.3.1.1 Controle em Malha aberta 39
2.3.1.2 Controle em malha fechada 41
2.4 MODULAÇÃO POR LARGURA DE PULSO 42
2.5 REGULADOR DE TENSÃO 43
2.6 FILTRO RLC 44
3 **DESENVOLVIMENTO** 45
3.1 HARDWARE 45
3.1.1 **Motor** 45
3.1.2 **Placa de controle** 47
3.1.2.1 Pinos Utilizados 49
3.1.3 **Placa de potência** 51
3.1.3.1 Regulador de tensão 52

3.1.3.2 Conversor de Nível CC 53
3.1.3.3 Transistor MOSFET de potência 55
3.1.3.4 Acelerador com potenciômetro 55
3.1.3.5 Sensor de Corrente 56
3.1.3.6 Pull up Encoder sensor hall do motor 56
3.1.3.7 Detector fonte 57
3.1.3.8 Circuito na matriz de contatos 58
3.1.3.9 Circuito de potência em placa auxiliar 61
3.1.3.10 Placa protótipo final 63
3.2 SOFTWARE DE CONTROLE 68
3.2.1 **Entradas digitais de ativação** 68
3.2.2 **Motor em funcionamento** 70
3.2.3 **Leitura e escrita via UART** 74
3.3 TESTES E VALIDAÇÕES 74
3.3.1 **Teste com software interrupção de timer** 74
3.3.2 **Teste de malha aberta** 76
3.3.3 **Teste de malha fechada** 80
4 **CONSIDERAÇÕES FINAIS** 85
REFERÊNCIAS 87
APÊNDICE A - CÓDIGO-FONTE TEMPORAL 90
APÊNDICE B - CÓDIGO-FONTE DE MALHA ABERTA 95
APÊNDICE C - CÓDIGO-FONTE DE MALHA FECHADA 105
ANEXO A - ESQUEMA DO CONTROLADOR DO MOTOR 115
ANEXO B - CÓDIGO-FONTE DE MALHA ABERTA 119
ANEXO C - CÓDIGO-FONTE DE MALHA FECHADA 122

1 **INTRODUÇÃO**

Observa-se uma inserção no mercado mundial do motor sem escovas com imãs permanentes, principalmente devido a suas diversas vantagens em relação aos modelos atuais de motores.

Porém, dentre muitas vantagens desse motor, a principal desvantagem trona-se o motivador de desenvolvimento desse trabalho, pois a necessidade de um controlador eletrônico complexo para operar o motor sem escovas com imãs permanentes traz uma demanda significativa ao mercado de pesquisa e desenvolvimento de soluções comercialmente viáveis e robustas, principalmente com a ascensão dos veículos híbridos e puramente elétricos previstos para a próxima década.

1.1 OBJETIVO GERAL

Desenvolvimento de um protótipo de controlador para motor sem escovas com imãs permanentes, baseado no microcontrolador dsPIC30F2010, apresentando as problemáticas da elaboração e os resultados dos testes de validação.

1.2 OBJETIVOS ESPECÍFICOS

Para tanto, esses estudos foram divididos em aspectos relevantes para o entendimento da dinâmica das técnicas de controle e suas interações com os aspectos físicos e eletroeletrônicos.

Desenvolvimento em matriz de contatos do circuito de potência.

Desenvolvimento de código fonte para controle do motor, baseado no microcontrolador dsPIC30F2010.

Elaboração de protótipo funcional para controle do motor.

Elaboração de testes de validação.

Apresentação dos resultados e futuras implementações.

2 **FUNDAMENTAÇÃO TEÓRICA**

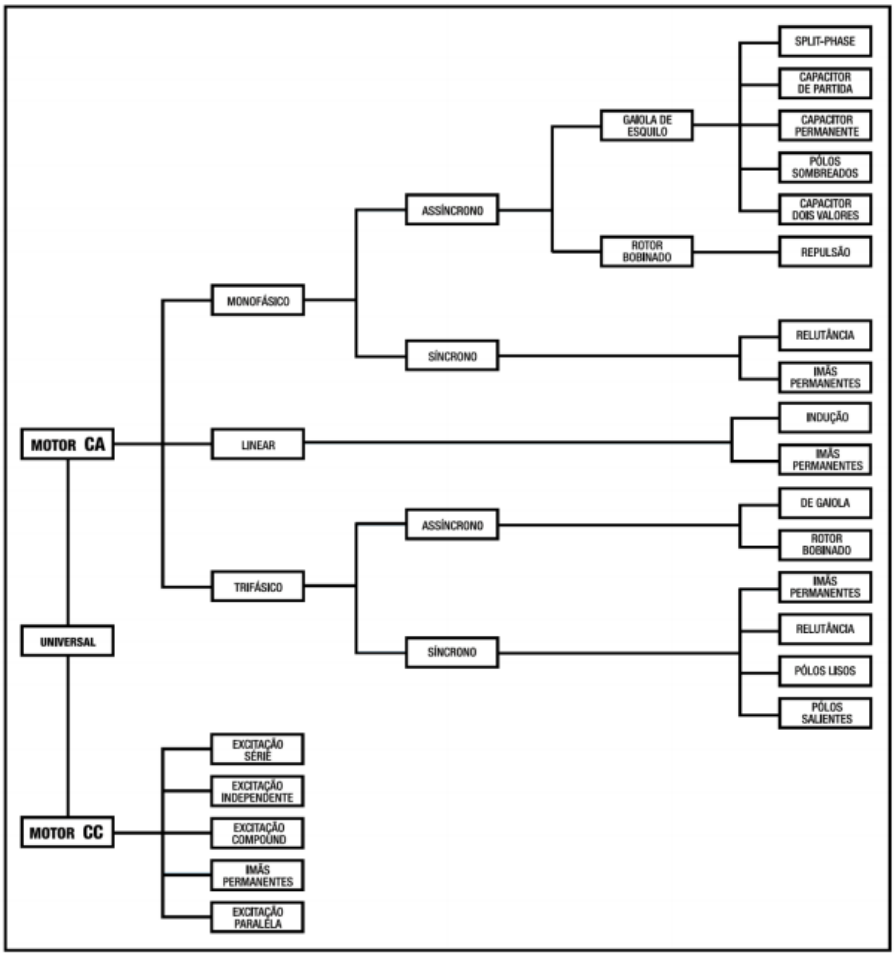
Este capítulo aborda os conceitos teóricos necessários para o desenvolvimento do trabalho.

2.1 MOTOR ELÉTRICO

Um motor é uma máquina que converte energia elétrica em energia mecânica de rotação. (GUSSOW, 2009, p. 249).

Os motores elétricos mais encontrados na indústria são arranjados conforme a *Figura 1.*

Figura 1 - Diagrama dos principais motores encontrados



Fonte: Adaptado de WEG (2016)

2.1.1 **MOTOR DE CORRENTE CONTÍNUA SEM ESCOVAS**

Do inglês *Brushless Direct Current* (BLDC), trata-se de um motor de corrente contínua que utilizada em seu rotor imãs permanentes ao invés de escovas, como existe nos motores de corrente contínua tradicionais. (MICROCHIP, 2003, com adaptações) .

Essa substituição altera em vários aspectos o motor BLDC, acarretando uma série de vantagens relevantes, e também algumas desvantagens. Na *Tabela 1* tem-se um comparativo entre as características do motor brushless em relação ao motor CC com escovas tradicional.

Tabela 1 - Comparando motor brushless com um motor CC com escovas

Característica	Motor BLDC	Motor CC com escovas
Comutação	Comutação eletrônica, baseada em sensores de posição Hall	Comutação por escovas
Manutenção	Baixa, pois não possui escovas que causam desgaste.	Manutenção periódica é necessária
Vida útil	Maior	Menor
Características de Velocidade / Torque	Plana - Possibilita operações com todas as velocidades com carga nominal	Moderadamente plana - Nas velocidades mais elevadas, fricção de escova aumenta, reduzindo assim o binário útil
Eficiência	Elevada - Sem queda de tensão nas escovas	Moderada
Potência de Saída / Tamanho	Alto - Tamanho reduzido devido a temperaturas características. Como o BLDC tem os enrolamentos no estator, que é conectado na carcaça, o calor têm uma dissipação melhor	Moderado / Baixo - O calor produzido pela armadura é dissipada no entreferro, aumentando assim a temperatura no entreferro e limitando especificações de tamanho / potência de saída
Inercia do motor	Baixa, porque tem ímãs permanentes no rotor. Isso melhora a resposta dinâmica	Maior inércia do rotor que limita as características dinâmicas
Range de Velocidade	Maior, nenhuma limitação mecânica de escovas / comutador	Baixa - Limitações mecânicas pelas escovas
Geração Ruído elétrico	Baixo	Arcos voltaicos nas escovas irão gerar ruído causando EMI nos equipamentos próximos
Custos para construção	Alto - Como possui ímãs permanentes, os custos de produção são mais elevados	Baixo
Controlador	Complexo e caro	Simples e barato
Requisitos do controlador	O controlador sempre é necessário para manter o motor em funcionamento. O mesmo controlador pode ser usado para controle de velocidade	Nenhum controlador é necessário para velocidades fixas. É necessário apenas se a velocidade variável for desejada

Fonte: Adaptado de MICROCHIP (2003, p. 7)

Devido a estas características, os motores BLDC são encontrados cada vez mais nas indústrias, assim como em seguimentos como eletrodomésticos, automotivos, aeroespacial, equipamentos médicos, equipamentos de automação industrial e instrumentação. (XIA, 2012, com adaptações).

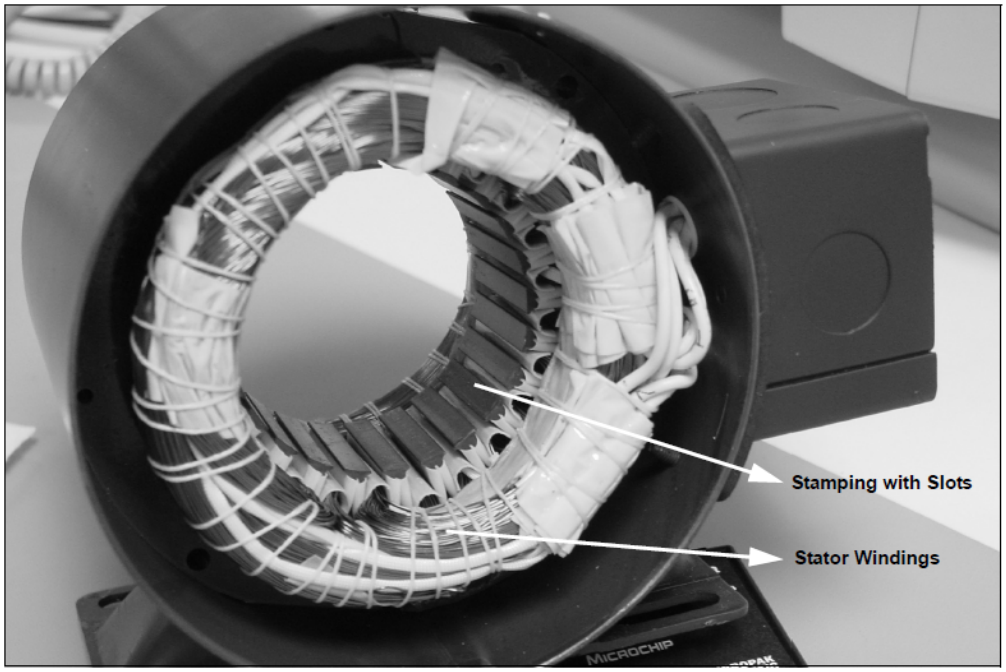
2.1.1.1 Construção e princípio de funcionamento

O motor BLDC é um motor síncrono, isso quer dizer que o campo magnético gerado pelo extator e o campo magnético gerado pelo rotor rotacionam na mesma frequência, não apresentando escorregamentos como comumente observa-se nos motores de indução e nos motores de corrente alternada não síncronos (MICROCHIP, 2003, com adaptações).

2.1.1.2 Estator

O estator de um motor BLDC consiste em laminados de aço empilhados com enrolamentos colocados nas ranhuras que são cortados axialmente ao longo da periferia interna como mostrado na *Figura 2.* Tradicionalmente, o estator assemelha-se ao de um motor de indução; Contudo, os enrolamentos são distribuídos de uma maneira diferente. A maioria dos motores BLDC tem três enrolamentos de estator conectados em forma de estrela. Cada um destes enrolamentos são construídos com numerosas bobinas interligadas para formar um enrolamento. Uma ou mais bobinas são colocadas nas ranhuras e são interligadas para fazer um enrolamento. Cada um destes enrolamentos é distribuído sobre a periferia do estator para formar um número par de pólos (MICROCHIP, 2003, p. 2).

Figura 2 - Estator de um motor BLDC



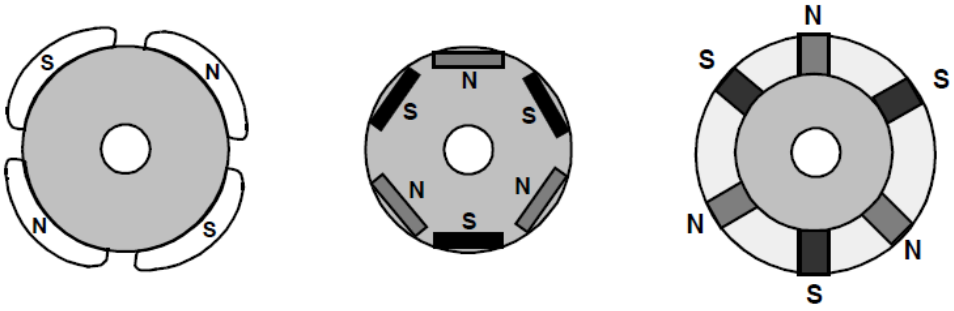
Fonte: MICROCHIP (2003, p. 3)

2.1.1.3 Rotor

O rotor é feito de ímã permanente e pode variar de dois a oito pares de pólos com pólos Norte (N) e Sul (S) alternados. Com base na densidade de campo magnético requerida no rotor, o material magnético apropriado é escolhido para fazer o rotor. Ímãs de ferrite são tradicionalmente usados para fazer ímãs permanentes. Como a tecnologia avança, ímãs de liga de terra rara estão ganhando popularidade. Os ímãs de ferrite são menos dispendiosos, mas têm a desvantagem de uma baixa densidade de fluxo para um dado volume. Em contraste, o material de liga tem uma densidade magnética elevada por volume e permite que o rotor se comprima mais para o mesmo binário. Além disso, estes ímãs de liga melhoram a relação tamanho-a-peso e dão maior torque para o motor do mesmo tamanho usando ímãs de ferrite (MICROCHIP, 2003, p. 4).

A *Figura 3* mostra secções transversais de arranjos diferentes de ímãs num rotor.

Figura 3 - Tipos de arranjos de rotores com imãs permanentes



Fonte: Adaptado de MICROCHIP (2003, p. 4)

2.1.1.4 Sensor de posicionamento

Os sensores de posição instalados no motor podem detectar a posição do rotor e transformá-lo em um sinal elétrico, fornecendo a informação de comutação correta para o circuito de chave lógica. Consequentemente, a comutação de corrente adequada dos enrolamentos é obtida de acordo com a informação de posição do rotor, e o rotor de imãs permanentes irá rodar continuamente devido ao campo magnético de rotação de passo gerado pela corrente no entreferro. (XIA, 2012, p. 30, com adaptações).

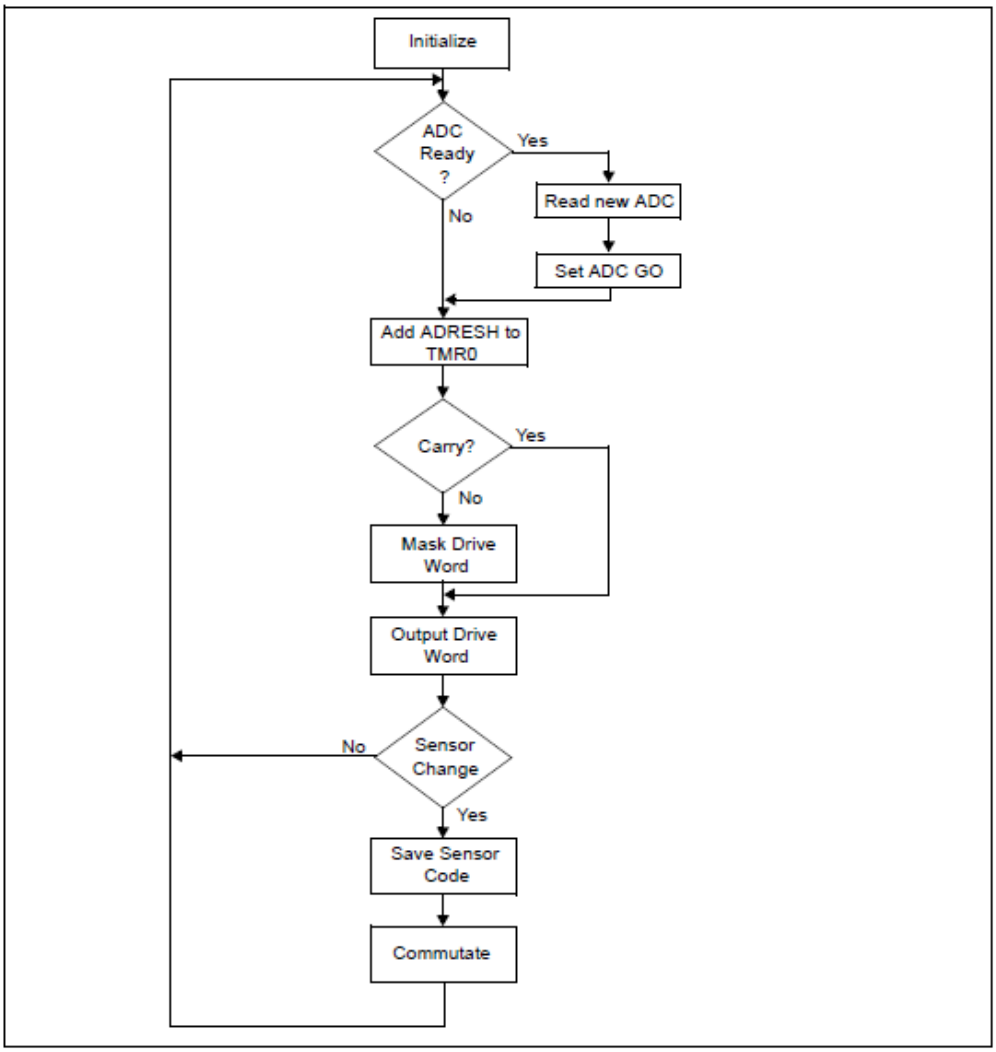
Existem vários tipos de sensores de posição e cada um tem suas próprias características. Atualmente, tem sido utilizada uma vasta gama de sensores electromagnéticos, fotoelétricos e magnéticos em motores BLDC. O sensor Hall, como um tipo de sensor magnético, tem as vantagens de volume compacto, baixo preço e operação conveniente. Portanto, é comumente usado em sistemas de controle de motores BLDC como o detector de posição do rotor (XIA, 2012, p. 30).

2.1.1.4.1 Sensor hall

Dispositivo capaz de detectar campos magnéticos presente em imãs, por exemplo, através do efeito hall. O Efeito Hall ocorre quando uma corrente em um condutor tem sua trajetória desviada pela ação de um campo magnético, o que faz com que a Tensão de Hall seja gerada, e esta pode ser aproveitada por um circuito externo, pois ela é proporcional à intensidade do campo que cria. No caso a criação dos sensores de campos magnéticos (INSTITUTO NEWTON C. BRAGA, com adaptações).

Em controle de motor BLDC realizados por microcontroladores da linha dsPIC, o sensoriamento através de técnicas pode ser relacionado com o fluxograma da *Figura 4.*

Figura 4 - Fluxograma da rotina de sensoriamento em Dspic



Fonte: MICROCHIP (2011)

2.1.1.5 Teoria de operação

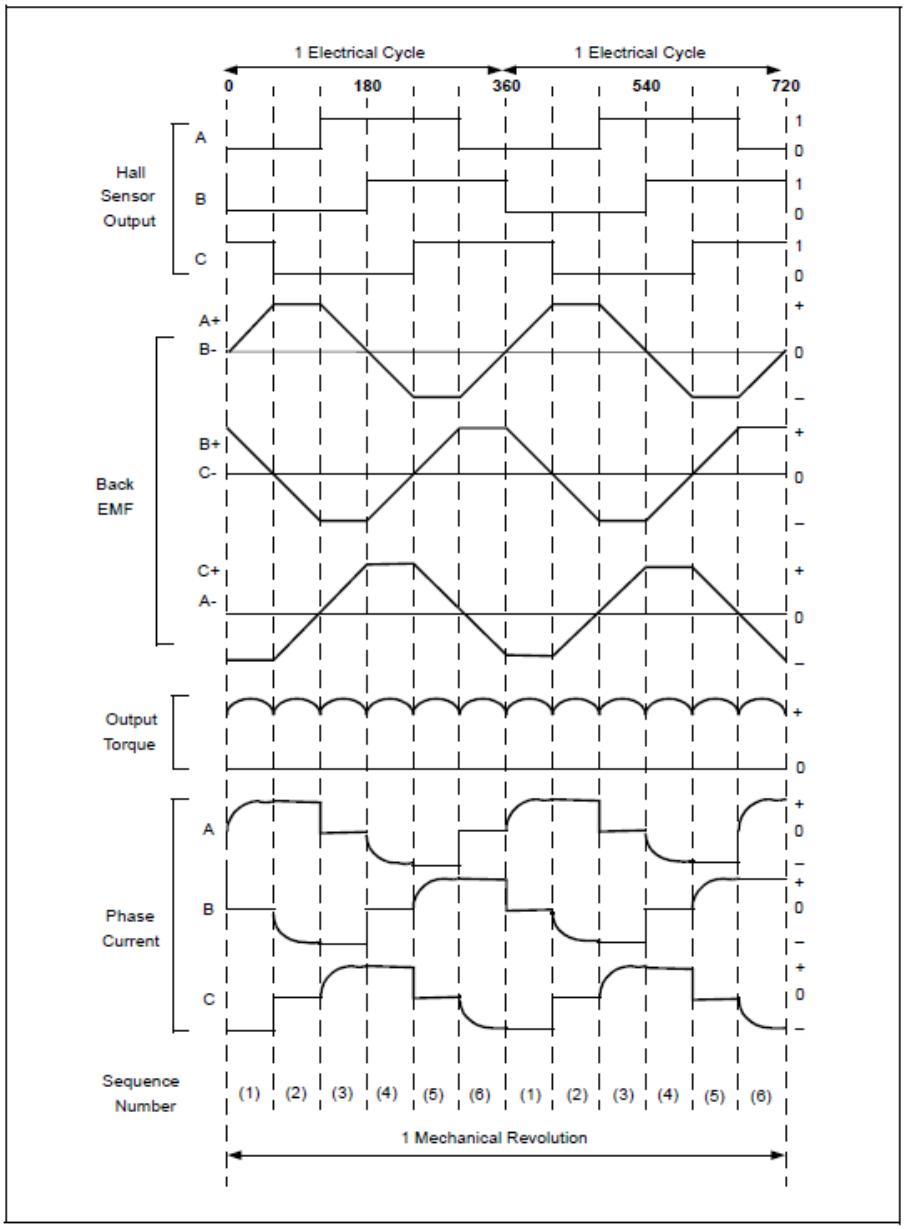
Cada sequência de comutação tem um dos enrolamentos energizados para potência positiva (a corrente entra no enrolamento); o segundo enrolamento é negativo (a corrente sai do enrolamento) e a terceira está em condições não energizadas. O torque é produzido por causa da interação entre o campo magnético gerado pelas bobinas do estator e os ímãs permanentes. Idealmente, o torque de pico ocorre quando estes dois campos estão a 90 ° uns dos outros e cai quando os campos se movem juntos. Para manter o motor em funcionamento, o campo magnético produzido pelos enrolamentos deve mudar de posição, à medida que o rotor se move para alcançar o campo do estator. O que é conhecido como "Comutação em Seis Passos" define a sequência de energia dos enrolamentos.

(MICROCHIP,2003)

2.1.1.5.1 Sequência de Comutação

A *Figura 5* mostra um exemplo de sinais do sensor hall em relação ao EMF de saída e à corrente de fase.

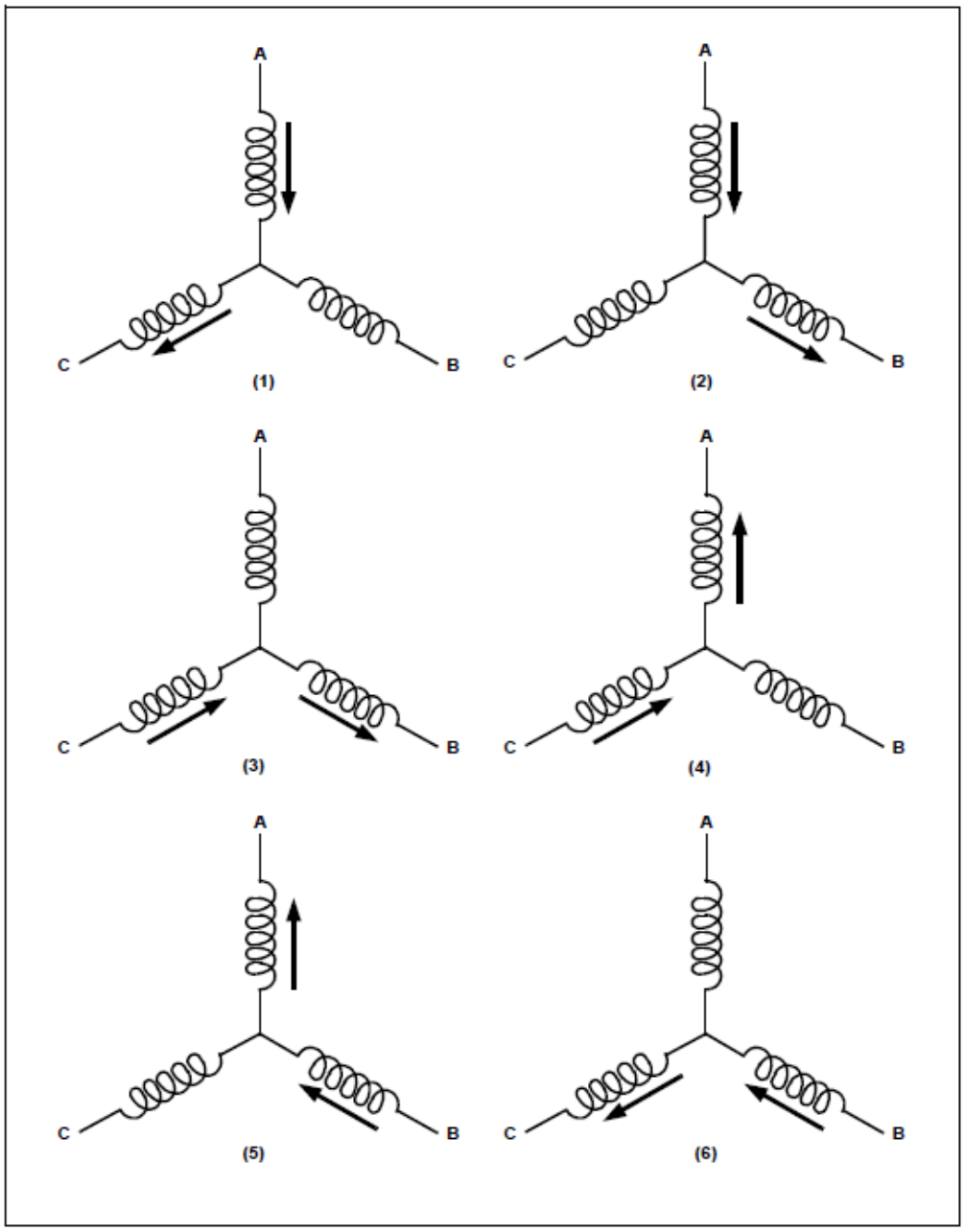
Figura 5 - Sinais de sensor hall, corrente de fase, EMF e torque de saída



Fonte: MICROCHIP (2003, p. 9)

Os números de sequência na *Figura 5* correspondem aos números dados na *Figura 6*.

Figura 6 - Sequência de energização dos enrolamentos



Fonte: MICROCHIP (2003, p. 10)

A *Figura 6* mostra a sequência de comutação que deve ser seguida em relação aos sensores Hall. Todos os 60 graus elétricos de rotação dos sensores Hall mudam o estado. Diante disto, são necessárias seis etapas para completar um ciclo elétrico. Em sincronia, com cada 60 graus elétricos, a mudança de corrente de fase deve ser atualizada. No entanto, um ciclo elétrico pode não corresponder a uma revolução mecânica completa do rotor. O número de ciclos elétricos a serem repetidos para completar uma rotação mecânica é determinado pelos pares de pólos do rotor. Para cada par de polos de rotor, um ciclo elétrico é completado. Assim, o número de ciclos / rotações elétricos é igual aos pares de polos do rotor. Com base

nas classificações de tensão e corrente do motor, esses Switch podem ser MOSFETs, ou IGBTs, ou transistores simples bipolares. A Tabela 2 e a Tabela 3 mostram a sequência em que esses interruptores de energia devem ser comutados com base nas entradas do sensor Hall, A, B e C. A Tabela 2 é para rotação no sentido horário do motor e a Tabela 3 é para rotação no sentido anti-horário. (MICROCHIP, 2003, com adaptações).

Tabela 2 - Sequência de rotação, sentido anti-horário

Sequence #	Hall Sensor Input			Active PWMs		Phase Current		
	A	B	C			A	B	C
1	0	1	1	PWM5(Q5)	PWM2(Q2)	Off	DC-	DC+
2	1	1	1	PWM1(Q1)	PWM2(Q2)	DC+	DC-	Off
3	1	1	0	PWM1(Q1)	PWM4(Q4)	DC+	Off	DC-
4	1	0	0	PWM3(Q3)	PWM4(Q4)	Off	DC+	DC-
5	0	0	0	PWM3(Q3)	PWM0(Q0)	DC-	DC+	Off
6	0	0	1	PWM5(Q5)	PWM0(Q0)	DC-	Off	DC+

Fonte: MICROCHIP (2003, p. 10)

Tabela 3 - Sequência de rotação, sentido horário

Sequence #	Hall Sensor Input			Active PWMs		Phase Current		
	A	B	C			A	B	C
1	0	0	1	PWM1(Q1)	PWM4(Q4)	DC+	Off	DC-
2	0	0	0	PWM1(Q1)	PWM2(Q2)	DC+	DC-	Off
3	1	0	0	PWM5(Q5)	PWM2(Q2)	Off	DC-	DC+
4	1	1	0	PWM5(Q5)	PWM0(Q0)	DC-	Off	DC+
5	1	1	1	PWM3(Q3)	PWM0(Q0)	DC-	DC+	Off
6	0	1	1	PWM3(Q3)	PWM4(Q4)	Off	DC+	DC-

Fonte: MICROCHIP (2003, p. 10)

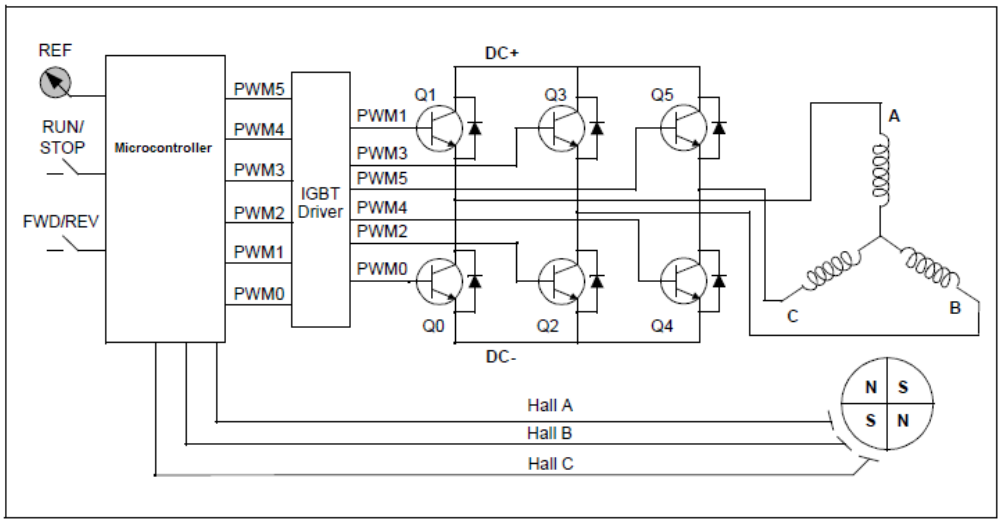
Este é um exemplo de sinais do sensor Hall com uma desvio de fase de 60 graus em relação um ao outro. Os sensores Hall podem estar a uma mudança de fase de 60 ° ou 120 ° entre si. Ao aplicar um controlador para um motor específico, a sequência definida pelo fabricante do motor deve ser seguida. Se os sinais marcados por PWM forem ligados ou desligados de acordo com a sequência, o motor funcionará na velocidade nominal. Isso pressupõe que a tensão do barramento CC é igual à tensão nominal do motor, além de qualquer perda em todos os interruptores. Para variar a velocidade, esses sinais devem ser modificados em largura de pulso (PWM) em uma freqüência muito maior do que a freqüência do motor. Como regra geral, a freqüência PWM deve ser pelo menos 10 vezes maior que a freqüência máxima do motor. Quando o ciclo de trabalho do PWM é variado dentro das sequências, a tensão média fornecida ao estator reduz, reduzindo assim a velocidade. (MICROCHIP, 2003, com adaptações).

Outra vantagem de ter o PWM é que, se a tensão do barramento CC for muito maior que a tensão nominal do motor, esta pode ser controlado limitando a porcentagem do ciclo de trabalho PWM correspondente ao da tensão nominal do motor. Isso adiciona flexibilidade ao controlador para conectar motores com diferentes tensões nominais e combinar a saída de tensão média pelo controlador, com a tensão nominal do motor, controlando o ciclo de trabalho PWM. Existem diferentes abordagens de controles. Se os sinais PWM forem limitados no microcontrolador, os interruptores superiores podem ser ativados por todo o tempo durante a sequência correspondente e o interruptor inferior correspondente pode ser controlado pelo ciclo de trabalho necessário no PWM. (MICROCHIP, 2003, com adaptações).

2.1.1.5.2 Controle de malha fechada

A Figura 7 apresenta um exemplo de diagrama de controlador utilizado em motor BLDC. A velocidade pode ser controlada em circuito fechado medindo a velocidade real do motor. O erro na velocidade ajustada e na velocidade real é calculado. Um controlador Proporcional, Integral e Derivativo (P.I.D.) pode ser usado para melhorar o erro de velocidade e ajustar dinamicamente o ciclo de trabalho PWM. Para requisitos de velocidade de baixo custo e baixa resolução, os sinais Hall podem ser usados para medir a resposta da velocidade. Um temporizador de um microcontrolador pode ser usado para contar entre duas transições. Com esta contagem, a velocidade real do motor pode ser calculada. Para medições de velocidade de alta resolução, um encoder óptico pode ser montado no motor, o que resulta em dois sinais com 90 graus de diferença de fase. Usando estes sinais, pode-se determinar a velocidade e a direção de rotação (MICROCHIP, 2003, com adaptações).

Figura 7 - Bloco diagrama controlador de motor BLDC.

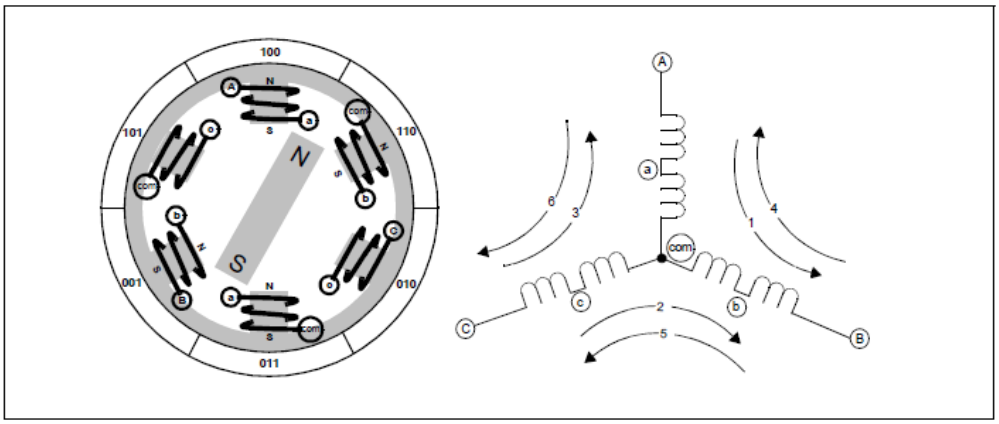


Fonte: Adaptado de MICROCHIP (2003, p. 10)

Além disso, a maioria dos codificadores dão um terceiro sinal de índice, que é um pulso por revolução. Isto pode ser usado para aplicações de posicionamento. Os codificadores ópticos estão disponíveis com diferentes opções de Pulse Per Revolution (PPR), que variam de centenas a milhares (MICROCHIP, 2003, com adaptações).

Na *Figura 8* é uma ilustração simplificada da construção do motor BLDC. A energia elétrica é convertida em energia mecânica pelas forças de atração magnética entre o rotor magnético permanente e um campo magnético rotativo induzido nos pólos do estator (MICROCHIP, 2011, com adaptações).

Figura 8 - Representação Motor BLDC simplificado.

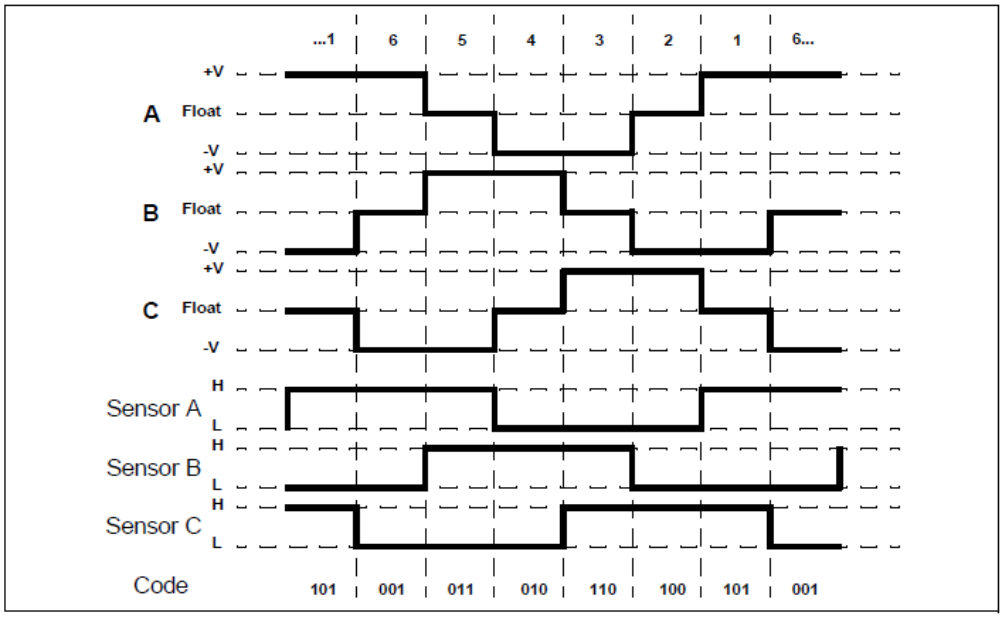


Fonte: MICROCHIP (2011)

Neste tipo de configuração de motor BLDC, existem três circuitos eletromagnéticos conectados em um ponto comum. Cada circuito eletromagnético é dividido no centro, permitindo assim que o rotor do ímã permanente se mova no

meio do campo magnético induzido. A maioria dos motores BLDC possui uma topologia de enrolamento trifásico com conexão em estrela. Um motor com esta topologia é conduzido por energização de duas fases por vez. O alinhamento estático mostrado na *Figura 9* é o que seria realizado através da criação de um fluxo de corrente elétrica do terminal A a B, observado como o caminho 1 no esquema na *Figura 8*. O rotor pode ser feito para girar no sentido horário 60 graus da A ao alinhamento B, alterando o caminho atual para o fluxo do terminal C para B, observado como o caminho 2 no esquema. O alinhamento magnético sugerido é usado apenas para fins de ilustração porque é fácil de visualizar. Na prática, o torque máximo é obtido quando o rotor de ímã permanente é de 90 graus de alinhamento com o campo magnético do estator (MICROCHIP, 2011, com adaptações).

Figura 9 - Representação da leitura do sensor contra o tempo de comutação



Fonte: MICROCHIP (2011)

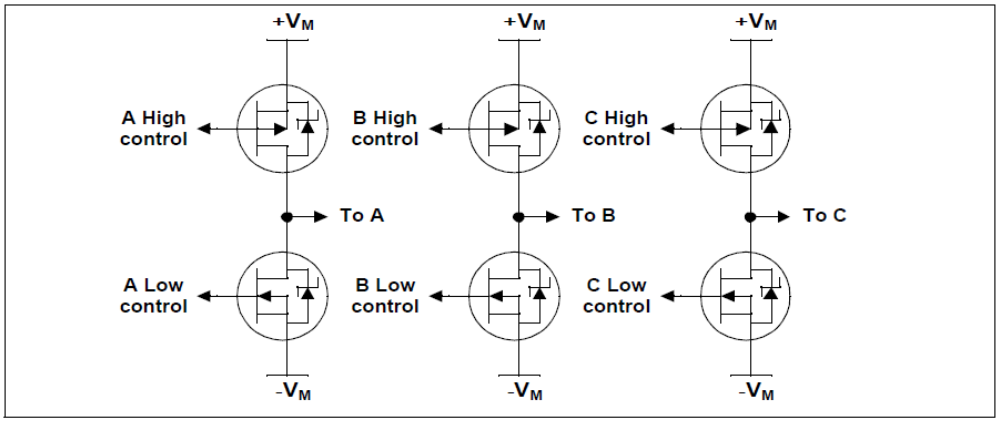
A chave para a comutação BLDC é detectar a posição do rotor e, em seguida, energizar as fases que produzirão a maior quantidade de torque. O rotor viaja 60 graus elétricos por etapa de comutação. O caminho de corrente do estator apropriado é ativado quando o rotor é de 120 graus do alinhamento com o campo magnético do estator correspondente e, em seguida, é desativado quando o rotor está a 60 graus do alinhamento, momento em que o próximo circuito é ativado e o processo se repete. A comutação para a posição do rotor, mostrada na *Figura 9*, seria na conclusão do caminho de corrente 2 e no início do caminho de corrente 3 para rotação no sentido horário. Comutando as conexões elétricas através das seis combinações possíveis, numeradas de 1 a 6, precisamente os momentos certos,

puxarão o rotor através de uma revolução elétrica. No motor simplificado da *Figura 8*, uma revolução elétrica é a mesma que uma revolução mecânica. Na prática, os motores BLDC possuem mais de um dos circuitos elétricos mostrados, conectados em paralelo um ao outro e um rotor magnético permanente multipolar correspondente. Para dois circuitos, existem duas revoluções elétricas por rotação mecânica. Então, para um motor de dois circuitos, cada fase de comutação elétrica cobriria 30 graus de rotação mecânica (MICROCHIP, 2011, com adaptações).

A maneira mais fácil de conhecer o momento correto para comutar as correntes de enrolamento é por meio de um sensor de posição. Muitos fabricantes de motores BLDC fornecem motores com um sensor de posição de efeito de Hall de três elementos. Cada elemento sensor emite um nível alto digital para 180 graus elétricos de rotação elétrica e um nível baixo para os outros 180 graus elétricos. Os três sensores são deslocados um do outro por 60 graus elétricos para que cada saída do sensor esteja alinhada com um dos circuitos eletromagnéticos. Um diagrama de temporização que mostra a relação entre as saídas do sensor e as tensões de transmissão do motor necessárias é mostrado na *Figura 9* (MICROCHIP, 2011, com adaptações).

Os números na parte superior da *Figura 9* correspondem às fases atuais mostradas na *Figura 8*. É evidente, a partir da *Figura 9,* que as três saídas do sensor se sobrepõem de forma a criar seis códigos únicos de três bits correspondentes a cada uma das fases de unidade . Os números mostrados em torno do periférico do diagrama do motor na *Figura 8* representam o código de posição do sensor. O pólo norte do rotor aponta para o código que é emitido naquela posição do rotor. Os números são os níveis de lógica do sensor onde o bit mais significativo é o sensor C e o bit menos significativo é o sensor A. Cada fase de unidade consiste em um terminal de motor conduzido alto, um terminal de motor baixo e um terminal de motor deixado flutuante. Um circuito de condução simplificado é mostrado na *Figura 10*. Os controles de acionamento individuais para drivers altos e baixos permitem alta movimentação, baixa movimentação e movimentação flutuante em cada terminal do motor. Uma precaução que deve ser tomada com este tipo de circuito de driver é que os drivers de lado alto e lado baixo nunca devem ser ativados ao mesmo tempo. Os resistores pull-up e pull-down devem ser colocados nas entradas do driver para garantir que os drivers estejam desligados imediatamente após um microcontrolador RESET, quando as saídas do microcontrolador são configuradas como entradas de alta impedância (MICROCHIP, 2011, com adaptações).

Figura 10 - Diagrama de circuito de saída com 3 fases



Fonte: MICROCHIP (2011)

Outra precaução contra ambos os drivers ativos ao mesmo tempo é chamada de controle de tempo morto. Quando uma saída troca do estado de nível alto para o estado de nível baixo, deve atender um tempo para o driver de nível alto ser desligada antes que próximo driver seja ativado. Os condutores demoram mais tempo para desligar do que para ligar. Então é necessário ter tempo extra para que ambos os condutores não estejam conduzindo ao mesmo tempo. Observa-se, na *Figura 9,* que o período de transição alto e o período de condução baixo de cada saída são separados por um período de fase de unidade flutuante. Este tempo morto é inerente ao cenário de unidade BLDC de três fases, portanto não é necessário um tempo especial para controle de tempo morto. A sequência de comutação BLDC nunca irá mudar o dispositivo de nível alto e o dispositivo de nível baixo em uma fase, ao mesmo tempo. (MICROCHIP, 2011, com adaptações).

2.2 MICROCONTROLADOR

Os microcontroladores, em geral, possuem todos os periféricos necessários em um único chip. Seu tamanho é muito pequeno, mesmo contendo muitos periféricos, tais como memórias, barramentos, contadores, portas de comunicação, conversores de sinais analógicos para digitais etc... Eles possuem desempenho menor que os microprocessadores, mas são ideias em aplicações que necessitam de menores dimensões, tempo e custos. (OLIVEIRA; ANDRADE, 2006, p. 34, com adaptações).

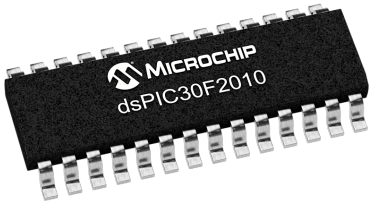
2.2.1 **DSPIC**

Os controladores de sinais dsPIC da Microchip são dispositivos que absorvem as principais características do mundo dos microcontroladores, unindo com a arquitetura voltada para o processamento em tempo real dos DSPs, assim criando uma linha de dispositivos intermediários (OLIVEIRA; ANDRADE, 2006, p. 36, com adaptações).

2.2.1.1 dsPIC30F2010

O microcontrolador dsPIC30F2010 foi desenvolvido com o propósito de atender as necessidades de controle de um motor BLDC, com módulos PWM e módulos de aquisição de sinal de encoders de alto desempenho (MICROCHIP, 2004, com adaptações).

Figura 11 - dsPIC30F2010 com encapsulamento SMD

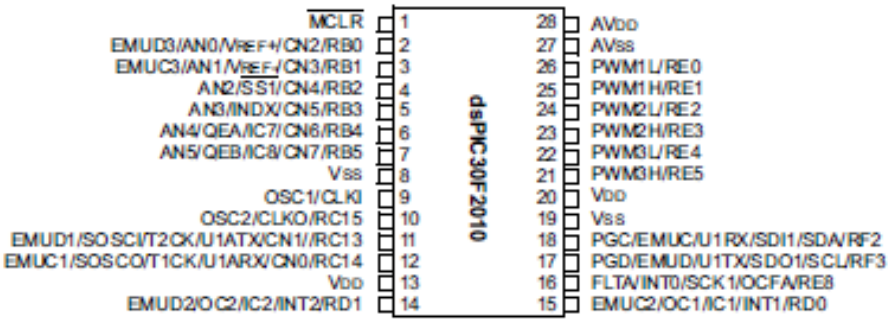


Fonte: MICROCHIP (2004)

A *Figura 11* apresenta o dsPIC30F2010 do tipo SMD, mas também é produzido do tipo thru-hole. (MICROCHIP, 2004, com adaptações).

Na Figura 12, o diagrama de pinos do encapsulamento SDIP é apresentado.

Figura 12 - Diagrama de pinos do dsPIC30f2010 encapsulamento SDIP



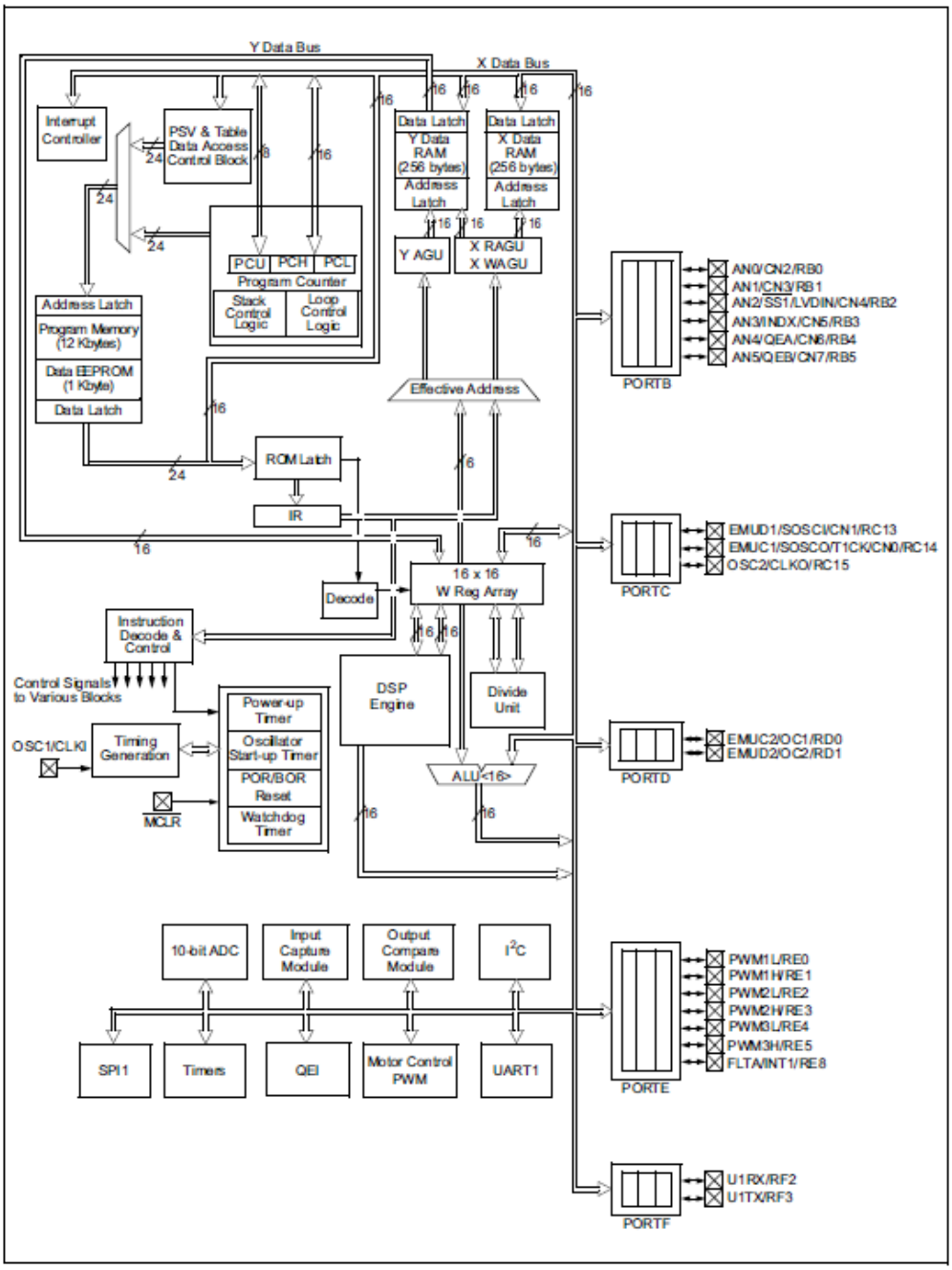
Fonte: MICROCHIP (2004, p. 3)

Tem capacidade de processar 30 MIPS (Milhões de instruções por segundo),

possue EEPROM interna, uma memória não volátil para armazenamento de dados (MICROCHIP, 2004, com adaptações).

A *Figura 13* apresenta o diagrama de blocos internos do dsPIC30F2010.

Figura 13 - Diagrama de blocos internos do dsPIC30F2010



Fonte: MICROCHIP (2004, p. 8)

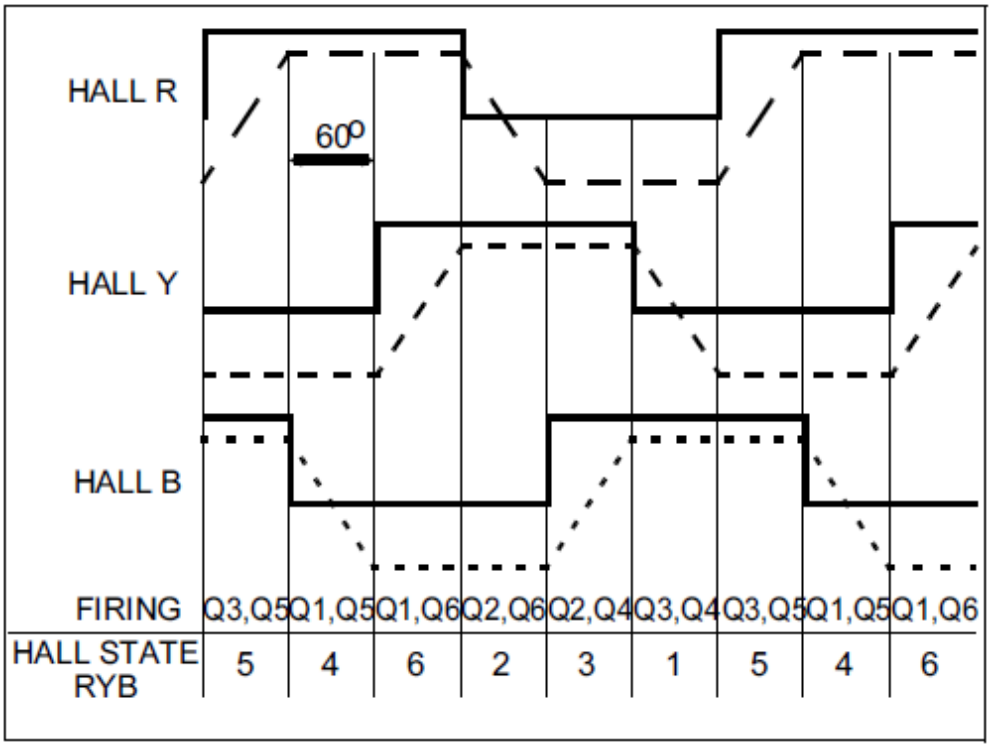
O dsPIC30F2010 têm como características principais, possuir um processador de alto desempenho com arquitetura RISC modificada (High-Performance Modified RISC CPU), possui arquitetura de conjunto de instruções otimizada para compilador C, Flexíveis modos de endereçamento, 84 instruções

básicas, 24-bit instruções de largura, 16-bit de dados largo caminho, até 12 Kbytes de espaço no chip programa Flash, 512 bytes de dados on-chip RAM,1 Kbyte não volátil de dados EEPROM, 16 x 16-bit matriz registro de trabalho, até 30 MIPs operação, até 27 fontes de interrupção sendo estas 8 níveis de prioridade selecionáveis pelo usuário, 3 fontes de interrupção externas e 4 utilizadas para o processador interceptar. (MICROCHIP, 2004, p. 2, com adaptações).

2.2.1.1.1 Técnicas sem sensor para a comutação do motor brushless

Os métodos discutidos aqui são aplicáveis somente a motores de 3 fases de construção padrão (sem bobinas de busca ou assimetrias deliberadas). Também é assumido que os blocos de energização convencionais de 120 ° são usados de tal forma que há períodos de tempo em que uma fase tem corrente zero fluindo e não está sendo utilizada ativamente. As fases conduzidas devem ser comutadas, ou comutadas a intervalos periódicos para executar o motor. Para permitir a comutação correta do motor, a posição absoluta dentro de um ciclo elétrico deve ser medida. Para a energização convencional, são necessárias seis comutações igualmente espaçadas por ciclo elétrico. Isso geralmente é implementado usando três interruptores de efeito Hall ou ópticos com um disco adequado no rotor. Não é necessária informação de posição contínua, apenas a detecção das instâncias de comutação requeridas. A *Figura 14* mostra as três saídas do sensor, juntamente com a correspondente forma de onda de voltagem EMF (BEMF) para cada fase (MICROCHIP, 2005, com adaptações).

Figura 14 - Diagrama de comutação BLDC



Fonte: MICROCHIP (2005)

Para detectar a posição do rotor monitorando uma propriedade do motor, claramente essa propriedade deve variar com a posição. Além disso, é desejável que a propriedade estabeleça uma posição única dentro de um ciclo elétrico, o que acrescenta robustez à técnica sem sensor. A variação na ligação de fluxo de fase com a posição produz torque. Este efeito pode ser dissecado em relutância e componentes BEMF, os quais podem variar com a atual, bem como a posição. O BEMF também varia linearmente com a velocidade. A variação da relutância ou BEMF pode ser monitorada diretamente, ou seu efeito em uma quantidade secundária pode ser usado em vez disso. (MICROCHIP, 2005, com adaptações)

A relutância representa o quão fácil é o fluxo fluir em torno do circuito magnético formado pelo aço, entreferro e ímãs. Os ímãs formam fontes de fluxo muito boas e são equivalentes a uma fonte de corrente elétrica. Enrolamentos de fase formam uma boa fonte de MMF e são equivalentes de fontes de tensão. Sob baixos níveis de carregamento magnético, o aço tem baixa relutância e é insaturado. Sob altos níveis de carga magnética (> 1,5 T tipicamente), a relutância do aço rapidamente começa a aumentar à medida que a saturação começa. O ar tem uma grande relutância, o que é independente do carregamento magnético. O material do imã se comporta de forma semelhante. Uma vez que a relutância varia de acordo com a posição, ela pode ser usada como base para a operação sem sensor. Em todos os motores BLDC, haverá alguma variação na relutância com o ângulo. Dos

terminais da máquina, a variação de relutância será evidente como uma variação na indutância. Esta variação tem a vantagem distinta de que a variação é detectável em velocidade zero. No entanto, o conhecimento prévio das características L (i, θ) do motor a ser controlado é necessário. Infelizmente, a variação de relutância com a posição é muito pequena para ser medida de forma confiável para muitos motores BLDC. Esta característica é especialmente verdadeira em motores com ímãs montados na superfície porque o ar-gap efetivo é grande. Como resultado, a parte dominante da relutância magnética é constante, fazendo assim qualquer variação residual com a posição difícil de medir. A variação de relutância também tende a ser baixa em motores que foram especificamente projetados para produção de ondulação de baixo torque, porque a relutância variável dá origem a um componente de torque adicional (MICROCHIP, 2005, com adaptações).

2.2.1.1.2 Técnicas com sensor para a comutação do motor brushless

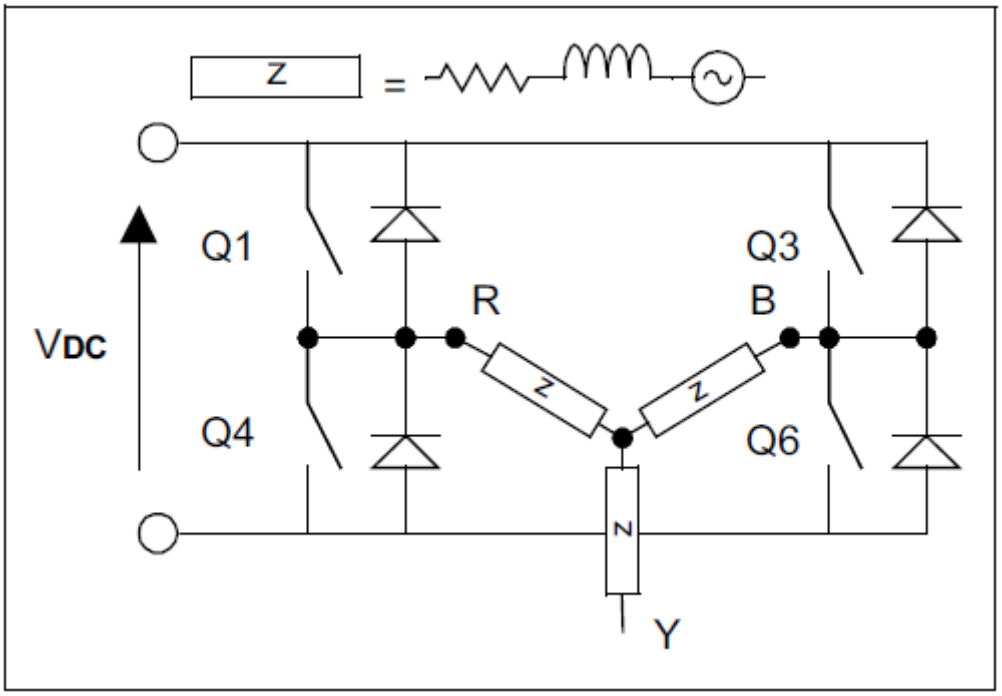
Este método é bastante fácil de implementar, mas para evitar que o rotor de ímã permanente fique bloqueado com o estator, a excitação no estator deve ser sequenciada de forma específica enquanto conhece a posição exata dos ímãs do rotor. A informação de posição pode ser obtida por um codificador de eixo ou, mais frequentemente, por sensores de efeito Hall que detectam a posição do ímã do rotor. Para um motor BLDC sensório trifásico típico, existem seis regiões ou setores distintos nos quais dois enrolamentos específicos estão excitados. Este método é bastante fácil de implementar, mas para evitar que o rotor de ímã permanente fique bloqueado com o estator, a excitação no estator deve ser sequenciada de forma específica enquanto conhece a posição exata dos ímãs do rotor. A informação de posição pode ser obtida por um codificador de eixo ou, mais frequentemente, por sensores de efeito Hall que detectam a posição do ímã do rotor. Para um motor BLDC sensório trifásico típico, existem seis regiões ou setores distintos nos quais dois enrolamentos específicos estão excitados. Estes são como mostrado na *Figura 14* (MICROCHIP, 2004, com adaptações).

Ao ler os sensores de efeito Hall, um código de 3 bits pode ser obtido com valores variando de 1 a 6. Cada valor de código representa um setor no qual o rotor está localizado atualmente. Cada valor de código, portanto, nos dá informações sobre quais enrolamentos precisam ser excitados. Assim, uma tabela de pesquisa simples pode ser usada pelo programa para determinar quais dois enrolamentos específicos para excitar e, assim, girar o rotor (MICROCHIP, 2004, com adaptações).

Usando esse método, pode-se obter a rotação de velocidade máxima do

motor BLDC. No entanto, para obter velocidade variável do motor BLDC, deve-se aplicar uma tensão variável aos terminais dos enrolamentos. Colocando isso em termos digitais, a voltagem variável pode ser obtida por diferentes ciclos de serviço de um sinal PWM, que vai aos enrolamentos do motor BLDC. O dsPIC30F2010 possui seis saídas PWM que podem ser conduzidas com o sinal PWM. Conforme mostrado na *Figura 15*, os três enrolamentos podem ser acionados em ON High, acionados ON Low ou não conduzidos, usando seis interruptores, IGBTs ou MOSFETs. Quando uma perna do enrolamento é conectada, por exemplo, ao lado alto, o sinal de ciclo de trabalho variável PWM pode ser injetado no driver do lado baixo. Isso tem o mesmo efeito que ter um sinal PWM no lado alto e conectar o lado baixo ao VSS ou GND (MICROCHIP, 2004, com adaptações).

Figura 15 - Bloco diagrama acionamento motor BLDC



Fonte: MICROCHIP (2004)

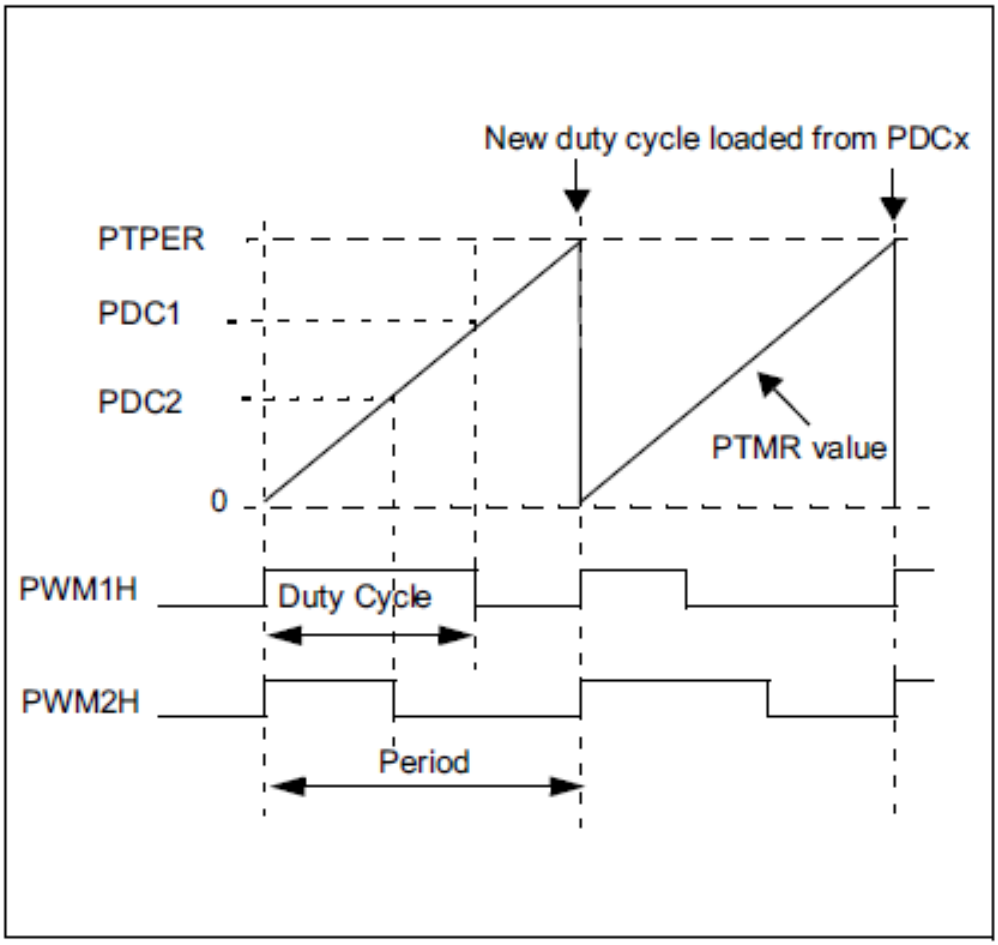
O PWM é fornecido pelo módulo dedicado de controle de motor do dsPIC30F2010. O módulo MCPWM foi projetado especificamente para aplicações de controle de motor (MICROCHIP, 2004).

O MCPWM possui um registro de base de tempo PTMR dedicado de 16 bits. Este temporizador é incrementado por uma variável definida pelo usuário, que pode ser tão baixo quanto o TCY. O usuário também decide o período necessário para o PWM, selecionando um valor e carregando nos registradores PTPER. O PTMR é comparado ao valor PTPER em cada TCY. Quando há uma partida, um novo período é iniciado. O ciclo de trabalho é controlado de forma semelhante,

carregando um valor nos três registros do ciclo de trabalho. Ao contrário do período, compara-se o valor no registro do ciclo de trabalho em cada intervalo TCY / 2 (ou seja, duas vezes mais rápido do que o período comparativo). Se houver uma correspondência entre o valor PTMR e o valor PDCx, a saída do ciclo de trabalho correspondente é conduzida baixa ou alta conforme ditado pelo modo PWM selecionado. As três saídas da comparação do ciclo de trabalho são canalizadas para um par de saída complementar onde uma saída é alta enquanto a outra é baixa e vice-versa. As duas saídas também podem ser configuradas como saídas independentes. Quando conduzido como saídas complementares, um tempo morto pode ser inserido entre o tempo em que o nível alto baixa e o nível baixo aumenta. Este tempo morto é configurado em hardware e tem um valor mínimo de TCY. A inserção de tempo morto impede a invasão inadvertida nos drivers de saída (MICROCHIP, 2004, com adaptações).

Existem vários modos nos quais o módulo MCPWM pode ser configurado. A saída alinhada é provavelmente o modo mais comum. A *Figura 16* ilustra a operação de um PWM alinhado por borda. No início do período, as saídas são todas elevadas. À medida que o PTMR aumenta, uma correspondência com os registros do ciclo de serviço faz com que a saída do ciclo de trabalho correspondente diminua, marcando assim o fim do ciclo de trabalho. A combinação PTMR com o registro PTPER causou um novo período para iniciar e todas as saídas são altas para iniciar um ciclo totalmente novo (MICROCHIP, 2004, com adaptações).

Figura 16 - PWM alinhado



Fonte: MICROCHIP (2004)

Os outros modos em que o MCPWM pode ser configurado são PWM de alinhamento central e PWM de disparo único. Esses modos não são discutidos porque não são tipicamente usados para controlar motores BLDC (MICROCHIP, 2004, com adaptações).

A característica importante do MCPWM usado nesta situação é o Override Control. O Override Control é a última etapa do módulo MCPWM. Permite ao usuário gravar diretamente no registro OVDCON e controlar os pinos de saída. O registro OVDCON tem dois campos de 6 bits nele. Cada um dos seis campos de bits corresponde a um pino de saída. A parte de alto byte do registro OVDCON, determina se o pino de saída correspondente é conduzido por um sinal PWM (quando definido como 1) ou (quando definido como 0) ativo / Inativo pelo campo de bits correspondente na parte de byte baixo do Registro OVDCON. Este recurso permite ao usuário ter sinais PWM disponíveis, mas não dirigindo, em todas as etapas de saída dos pinos. Para motores BLDC, o mesmo valor é escrito em todos os registros PDCx (MICROCHIP, 2004, com adaptações).

Dependendo do valor no registro OVDCON, o usuário pode selecionar qual

pino obtém o sinal PWM e qual pin é ativado ou inativo. Ao controlar o motor BLDC, é necessário excitar dois pares de enrolamentos dependendo de onde o rotor está localizado e ditado pelo valor dos sensores do corredor. Na rotina do Serviço de Interrupção CN, os sensores Hall são lidos e, em seguida, o valor dos sensores é usado como um deslocamento em uma tabela de pesquisa que corresponde ao valor que será carregado no registro OVDCON. A *Tabela 4* e a *Figura 17* mostram como os valores diferentes são carregados no registro OVDCON dependendo de qual setor o rotor está localizado e, assim, quais enrolamentos precisam ser excitados (MICROCHIP, 2004, com adaptações).

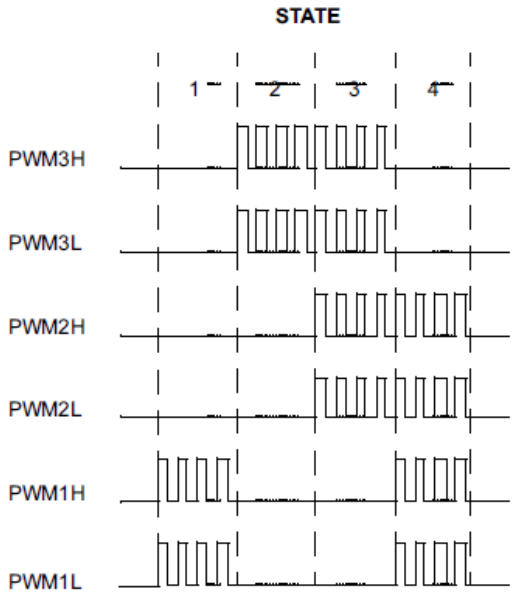
Tabela 4 - Exemplo de tabela OVDCON

State	OVDCON<15:8>	OVDCON<7:0>
1	00000011b	00000000b
2	00110000b	00000000b
3	00111100b	00000000b
4	00001111b	00000000b

Fonte: MICROCHIP (2004)

A partir da *Tabela 4*, obtêm-se a resposta do sinal PWM.

Figura 17 - Exemplo de saida PWM em relação ao OVDCON

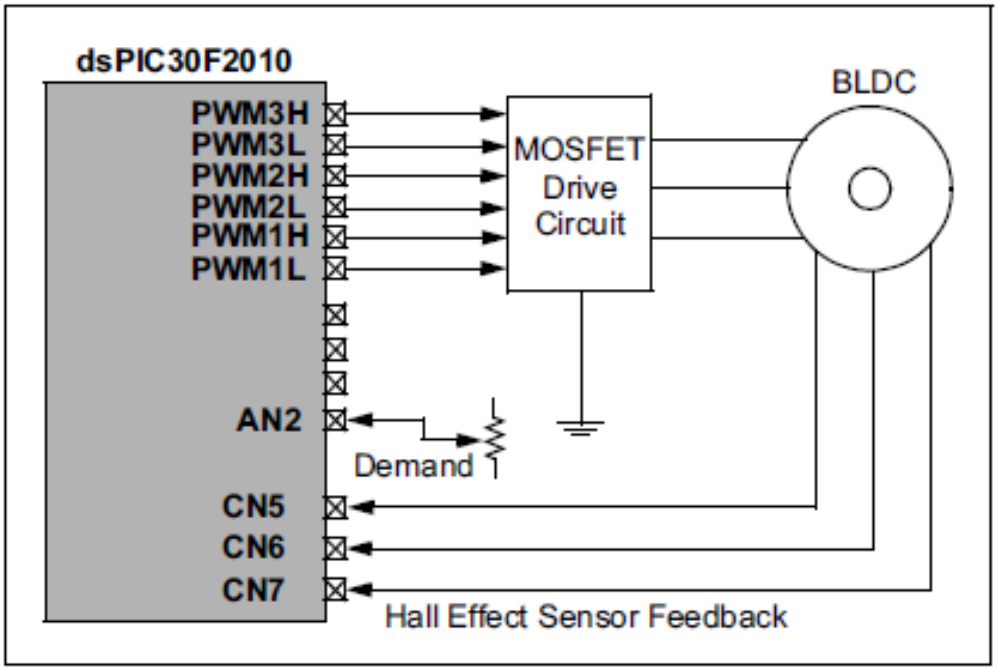


Fonte: MICROCHIP (2004)

2.2.1.1.3 Descrição de hardware para driver específico

O diagrama de blocos da *Figura 18* mostra como o motor BLDC é conduzido usando um dsPIC30F2010. Para um esquema detalhado, consulte o Anexo A.

Figura 18 - Diagrama de bloco do microcontrolador



Fonte: MICROCHIP (2004)

As seis saídas MCPWM estão conectadas a três pares de driver MOSFET (IR2101S), que, por sua vez, estão conectados a seis MOSFETs (IRFR2407). Estes MOSFETs estão conectados em um formato de ponte trifásica aos três enrolamentos do motor BLDC. Na implementação atual, a tensão MOSFET máxima é de 70 Volts e a corrente MOSFET máxima é de 18 Amps. É importante notar que uma dissipação de calor adequada deve ser fornecida se as capacidades máximas estiverem sendo utilizadas. Os drivers MOSFET também exigem uma maior tensão (15V) para operar, portanto, este nível de tensão precisa ser fornecido. O motor é um motor BLDC de 24V, de modo que a tensão de barramento DC + para CC é de 24V. Um 5V regulado é fornecido para conduzir o dsPIC30F2010. As três entradas do sensor de efeito Hall são conectadas aos pinos de entrada que possuem circuitos de notificação de mudança associados a eles. Essas entradas são ativadas juntamente com a interrupção. Se uma mudança ocorrer em qualquer um desses três pinos, uma interrupção é gerada. Para fornecer uma demanda de velocidade, um potenciômetro é conectado a uma entrada ADC (RB2). Para iniciar e parar o motor, um interruptor de botão é fornecido no RC14. Para fornecer algum feedback

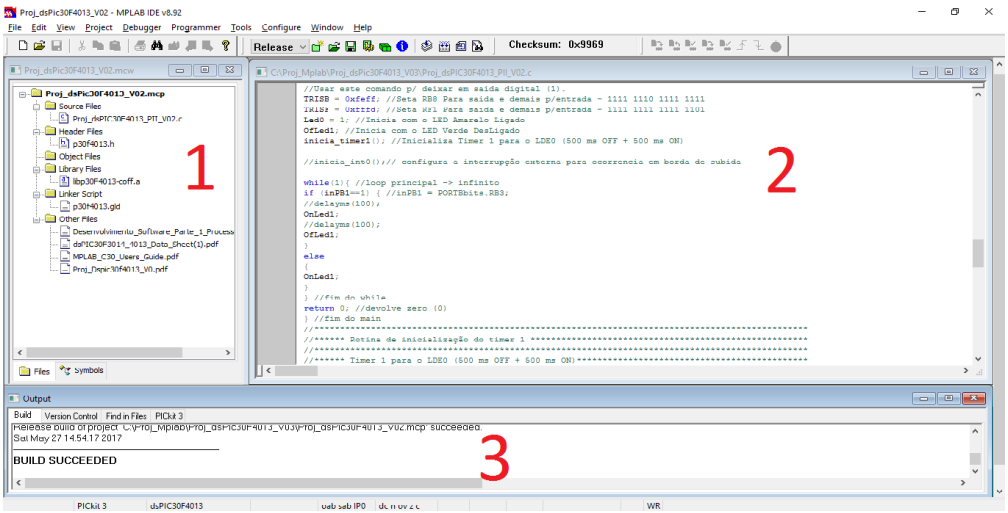
atual para o motor, um resistor de baixo valor (25 miliohms) está conectado entre a tensão do barramento DC e terra ou Vss. A tensão gerada por este resistor é amplificada por um amplificador operacional externo (MCP6002) e alimentada a uma entrada ADC (RB1).

2.2.1.2 MPLAB IDE

O MPLAB IDE é um ambiente de desenvolvimento fornecido pela Microchip Technology, fabricante dos micro controlador dsPIC30F2010, que, juntamente com o Compilador C30, torna-se uma ferramenta poderosa para programação desses microcontroladores utilizando a Linguagem C, visando uma interface moderna e intuitiva, com recursos indispensáveis para o programador usufruir dos recursos de seu hardware de maneira otimizada. Cada modelo de microcontrolador possui características diferentes, portas diferentes com seus respectivos recursos. Deve-se lembrar que em cada projeto deve-se incluir o arquivo de biblioteca e de ligação específicos para o microcontrolador escolhido para que o compilador possa funcionar corretamente (OLIVEIRA; ANDRADE, 2006, com adaptações).

A *Figura 19* mostra uma tela capturada de um projeto em andamento utilizando o ambiente MPLAB IDE.

Figura 19 - Ambiente de desenvolvimento MPLAB IDE



Fonte: O autor (2017)

Na janela 1, visualiza-se a organização dos arquivos que compõem o projeto. Na janela 2, o código a ser desenvolvido com a linguagem C e na janela 3, mensagens de saída ao efetuarmos as validações e compilação de nosso programa, onde, em caso de erro na escrita da sintaxes do programa, mensagens de alerta e de falha de compilação nos indicam a provável linha de código errônea (OLIVEIRA;

ANDRADE, 2006, com adaptações).

2.2.1.3 Pickit 3

Integrado a plataforma MPLAB IDE, o Pickit 3 trata-se de um dos hardware utilizados pela fabricante MICROCHIP no processo de gravação do código desenvolvido para o microcontroladores. Sendo integrado a plataforma MPLAB IDE, conta com otimizações que facilitam ao usuário sua utilização, agregando agilidade ao desenvolvimento de projetos (MICROCHIP, 2010, com adaptações).

Este é conectado via USB ao Computador do Usuário, onde é reconhecido e instalado automaticamente pelo MPLAB IDE e ainda, se o microcontrolador estiver conectado à porta de programação corretamente e atender a todos os pré-requisitos de gravação, imediatamente já é possível efetuar a programação deste. O programador é ilustrado através da foto na *Figura 20* (MICROCHIP, 2010, com adaptações).

Figura 20 - Programador Pickit 3



Fonte: O autor (2017)

2.3 LINGUAGEM C

Uma linguagem de programação muito difundida atualmente, principalmente devido a sua característica de escrever softwares otimizados para o processador como o assembly, mas também possui uma série de procedimentos prontos, como

nas linguagens de alto níveis (OLIVEIRA; ANDRADE, 2006, p. 74).

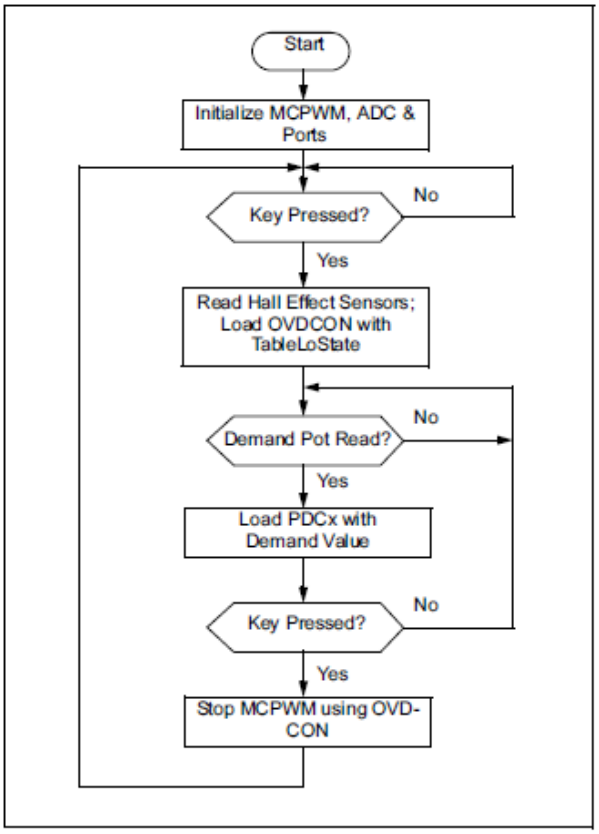
2.3.1 **DESCRIÇÃO FIRMWARE**

Dois programas de firmware estão incluídos no Anexo B e no Anexo C para ilustrar os métodos de controle no dsPIC30F2010. Um programa usa controle de velocidade em malha aberta. O outro usa realimentação proporcional e integral para controle de velocidade em malha fechada. O método de malha aberta geralmente não é prático para aplicações reais. Ele é incluído aqui principalmente para ilustrar a metodologia da unidade de motor BLDC (MICROCHIP, 2004, com adaptações).

2.3.1.1 Controle em Malha aberta

No controle de malha aberta, o MCPWM controla diretamente a velocidade do motor com base na entrada de tensão do potenciômetro de velocidade. Depois de inicializar as entradas MCPWM, ADC, Portas e Notificação de Mudança, o programa espera um sinal de ativação (por exemplo, uma tecla pressionada) para indicar um começo (conforme a *Figura 21*). Quando a tecla é pressionada, os sensores Hall são lidos. Com base no seu valor, um valor correspondente é recuperado da tabela e escrito no OVDCON. Neste ponto, o motor começa a girar (MICROCHIP, 2004, com adaptações).

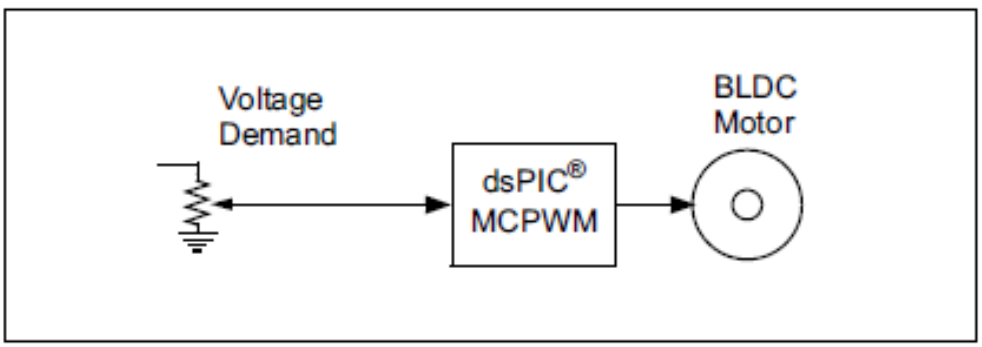
Figura 21 - Fluxograma de inicialização do controle de malha aberta



Fonte: MICROCHIP (2004)

Em primeiro lugar, o valor do ciclo de trabalho é mantido em um padrão de 50%. No próximo ciclo do programa principal, no entanto, o potenciômetro é lido e seu valor (isto é, o valor correto da demanda) é inserido como o ciclo de trabalho. Isso determina a velocidade do motor. Quanto maior o valor do ciclo de trabalho, mais rápido o motor girará. A velocidade é controlada pelo potenciômetro de controle de tensão, conforme mostra-se na *Figura 22* (MICROCHIP, 2004, com adaptações).

Figura 22 - Esquemático controle de malha aberta

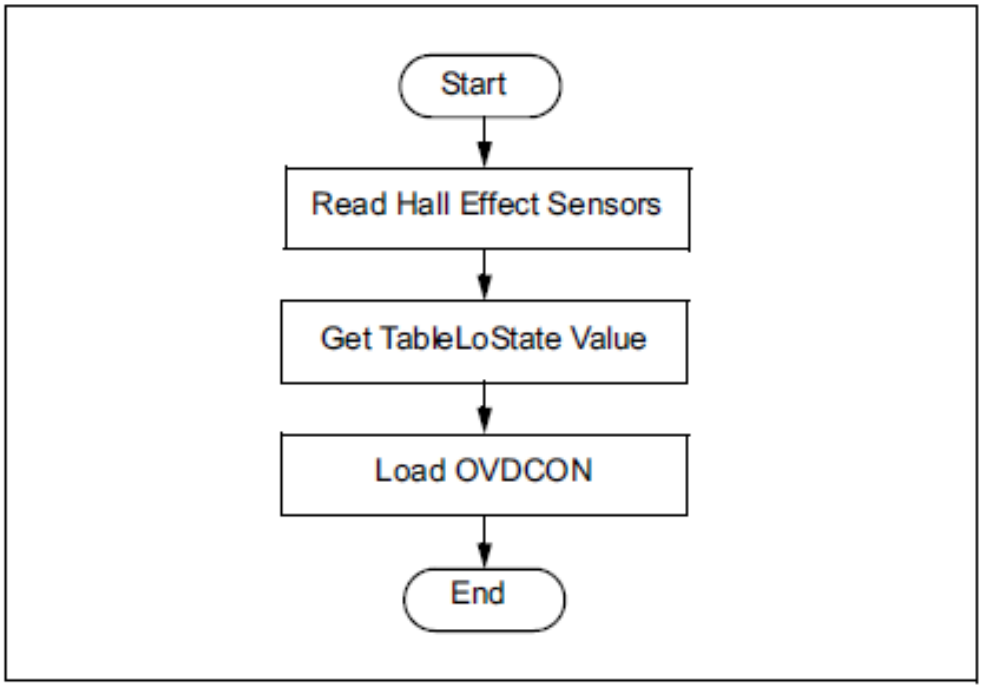


Fonte: MICROCHIP (2004)

Os sensores de efeito Hall estão conectados ao pino de Notificação de

Mudança. A interrupção CN está habilitada. À medida que gira o rotor, a posição do ímã do rotor muda e o rotor entra em um setor diferente. Cada nova posição é sinalizada por uma Interrupção CN. Na rotina de Interrupção CN, apresenta a *Figura 23*, os sensores de efeito Hall são lidos e com base no valor, um valor de pesquisa de tabela é obtido e gravado no registro OVDCON. Esta ação assegurará que os enrolamentos corretos estejam excitados no setor certo e o motor continuará girando (MICROCHIP, 2004, com adaptações).

Figura 23 - Fluxograma loop principal

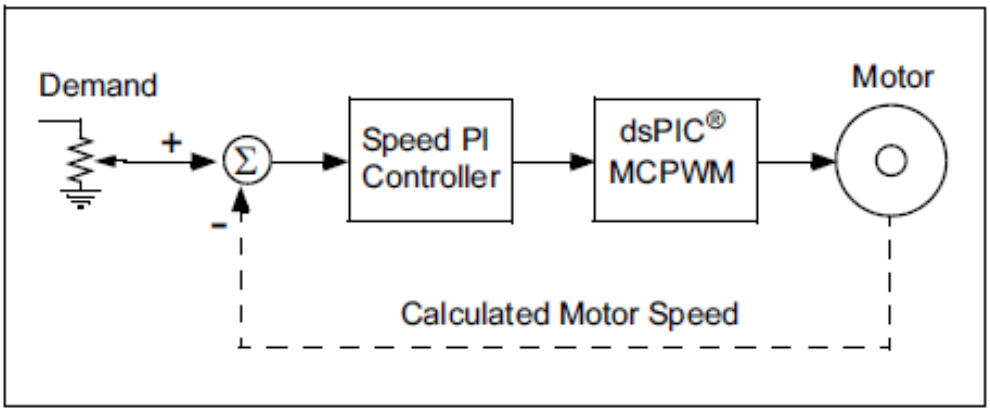


Fonte: MICROCHIP (2004)

2.3.1.2 Controle em malha fechada

Na versão de controle em circuito fechado do firmware, a principal diferença é que o pote é usado para configurar a demanda. O controle fornece controle proporcional e integral (PI) da velocidade. Para medir a velocidade real, TMR3 é usado como um temporizador para abrir um ciclo elétrico completo. Uma vez que se está usando um motor de dez polos, cinco ciclos elétricos resultam em um ciclo mecânico. Se T segundos for o tempo de um ciclo elétrico, então a velocidade S = 60 / (P / 2 * T) rpm, onde P é o número de polos do motor. A *Figura 24* mostra um diagrama de bloco com controle de malha fechada. (MICROCHIP, 2004, com adaptações).

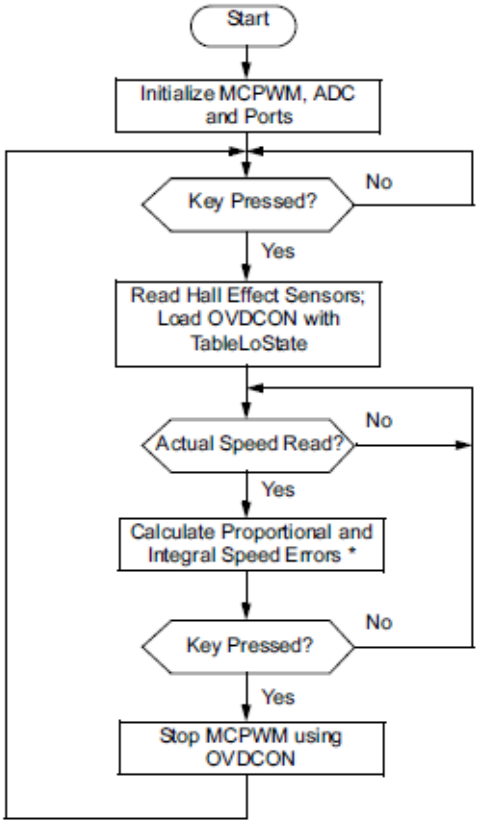
Figura 24 - Diagrama de bloco de malha fechada



Fonte: MICROCHIP (2005)

Um fluxograma de controle em circuito fechado é mostrado na *Figura 25*.

Figura 25 - Fluxograma da inicialização de programa em malha fechada



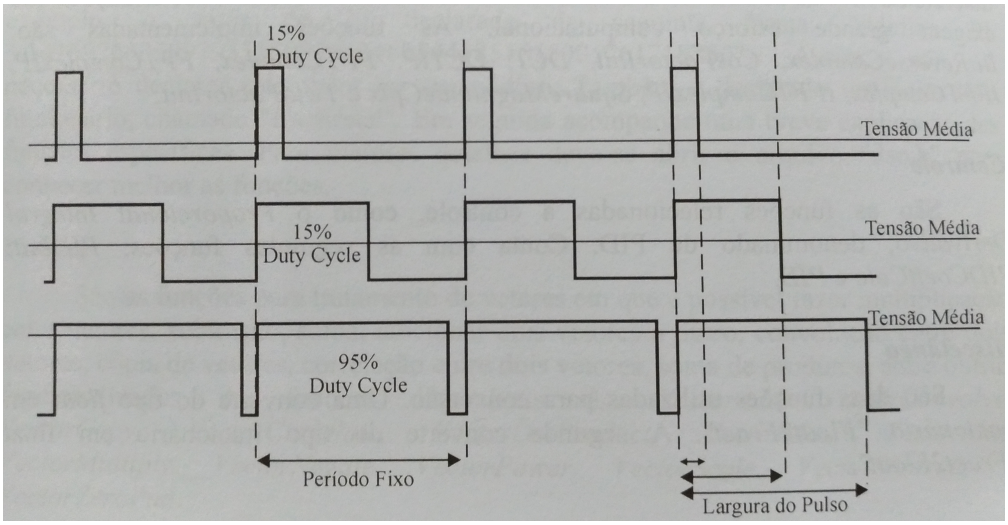
Fonte: MICROCHIP (2004)

2.4 MODULAÇÃO POR LARGURA DE PULSO

Modulação por Largura de pulso é um poderoso recurso, muito utilizado para o controle de motores. A partir dele, é possível gerar um sinal analógico (tensão variável), apensar de a saída ser um sinal digital que assume apenas os níveis lógicos: alto e baixo. A saída gerada é uma onda quadrada, como a frequência constante e uma largura do pulso variável. Esses conceitos estão diretamente relacionados com o período fixo e o Duty cycle respectivamente (OLIVEIRA; ANDRADE, 2006, com adaptações).

O ciclo de trabalho (Duty Cycle) define o tempo de sinal ativo (nível lógico alto) em um período fixo. Assim, quando tem-se um *Duty cycle* de 100%, tem-se nível lógico alto por todo o período e quando tem-se 0%, tem-se nível lógico baixo por todo o período. Um *Duty cycle* de 50% define a metade do período em nível lógico alto e a outra metade em nível lógico baixo, dando uma tensão média. Se operarmos em níveis TTL, em que a tensão em nível alto é 5 V, esse ciclo ativo fornece uma tensão média de 2,5 V. Esses conceitos são demonstrados na *Figura 26. (OLIVEIRA; ANDRADE, 2006, com adaptações).*

Figura 26 - Sinais de Modulação por Largura de Pulso



Fonte: OLIVEIRA e ANDRADE (2006)

2.5 REGULADOR DE TENSÃO

Os reguladores de tensão na forma de circuitos integrados de três terminais

são quase que obrigatórios em projetos de fontes de alimentação para circuitos de pequena e média potência. Os tipos da série 7800 que podem fornecer tensões de 5 a 24 volts tipicamente com corrente de 1 A são extremamente atraentes para projetos e que são apresentados em invólucro TO-220 (INSTITUTO NEWTON C. BRAGA)

A tensão máxima de entrada para os tipos de 5 à 18 volts é de 35 V. Para o tipo de 24 V a tensão de entrada máxima é de 40 V; da tensão máxima de entrada para os tipos de 5 à 18 volts é de 35 V. Para o tipo de 24 V a tensão de entrada máxima é de 40 V; de qualquer modo, para um bom funcionamento a tensão de entrada deve ser no mínimo 2 V mais alta que a tensão que se deseja na saída (INSTITUTO NEWTON C. BRAGA).

2.6 FILTRO RLC

Pelo seu comportamento elétrico, os indutores e os capacitores podem ser usados em muitos circuitos que envolvam sinais de determinadas frequências. Dentre esses circuitos, podemos citar os filtros, capazes de permitir a passagem de sinais de determinadas frequências ou bloquear sinais de determinas frequências. (INSTITUTO NEWTON C BRAGA).

Os capacitores oferecem uma pequena oposição aos sinais de altas frequências enquanto os indutores oferecem uma pequena oposição aos sinais de baixas frequências. combinando esses dois componentes, temos então o que denominamos "filtros", e, basicamente, podemos ter os seguintes tipos de filtros: passa-baixas, que oferecem pouca oposição aos sinais de baixas frequências mas que bloqueiam os sinais de altas frequências; passa-altas, que oferecem forte oposição aos sinais de baixas frequências, mas que deixam passar os sinais de altas frequências.

Passa-faixas ou Passa-Bandas, que deixam passar com pouca oposição os sinais de uma certa faixa de freqüência, mas que bloqueiam os sinais que estejam foram dela.

Rejeita-Faixa ou Rejeita-Banda que bloqueiam os sinais que estão dentro de uma certa faixa de freqüência, mas deixam passar, com pouca oposição, os que estão fora dela (INSTITUTO NEWTON C BRAGA, com adaptações).

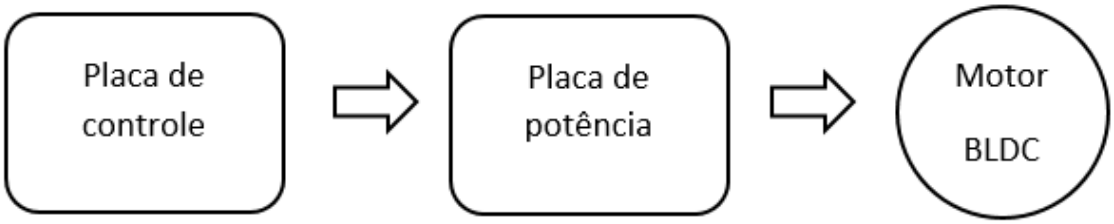
3 **DESENVOLVIMENTO**

A realização desse trabalho acontece em duas etapas, o desenvolvimento hardware, com relatos do problemas e as soluções encontradas no desenvolvimento das placas protótipos, e, posteriormente, de software, relata-se as principais rotinas desenvolvidas até a implementação do objetivo principal.

3.1 HARDWARE

Desenvolve-se o hardware conforme o diagrama de blocos da *Figura 27*.

Figura 27 - Diagrama de blocos do hardware



Fonte: O autor (2017)

Atenta-se na *Figura 27,* que a placa protótipo é dividida em placa de controle e placa de potência, o que proporciona agilidade no desenvolvimento do projeto, pois gera a possibilidade da utilização de uma placa de controle já desenvolvida em projetos anteriores pela Universidade Feevale.

3.1.1 **Motor**

O motor de imãs permanentes a ser controlado tem como fabricante a LIN ENGINEERING, na *Tabela 5*, tem-se as características principais:

Tabela 5 - Especificações motor brushless

Característica	Valor
Número de polos do rotor	4 pólos
Inércia do rotor	0,95 oz-in^2
Tensão nominal	34 VDC
Potência nominal	175,82 W
Torque nominal	54 oz-in
Corrente Nominal	6,72 A
Velocidade nominal	4400 RPM
Constante de torque nominal	8,96 oz-in
Sensoramento	Sensor Hall

Fonte: Adaptado de ALLTRONICS (2013)

Conforme a *Tabela 5*, o motor conta com características adequadas para o desenvolvimento de um módulo de potência sem a necessidade de componentes de correntes e tensões elevadas, e inclui em seu interior sensores Hall para o controle de posição; e mesmo que a tensão nominal do motor de 34 V, no projeto trabalha-se com uma tensão máxima de 24 V.

Na *Figura 28* tem-se o visual do motor.

Figura 28 - Motor no.: CO-BL23B46-02-02RO



Fonte: ALLTRONICS (2013)

Nota-se, na *Figura 28,* um conector contempla todos as conexões do motor, sendo esse conector de modelo *MSTB 2.5* de 8 vias apresentados na *Tabela 6*.

Tabela 6 - Diagrama do conector do motor sem escovas

Pino	Cor	Descrição
1	Amarelo	Fase A
2	Vermelho	Fase B
3	Preto	Fase C
4	Preto e Branco	Hall Ground
5	Azul	Hall A
6	Verde	Hall B
7	Branco	Hall C
8	Vermelho e Branco	Hall VCC 5 V

Fonte: Adaptado de ALLTRONICS (2013)

Um detalhe importante trata-se da tensão de alimentação do Hall ser muito inferior a tensão nominal do motor, sendo necessária uma fonte de alimentação de diferente tensão para fazer uso do sensor.

Ainda como característica do motor sem escovas, tem-se a sequência de comutação dada pelos sensores hall, essa sequência caracteriza a ordem de eventos do chaveamento das fases do motor a fim de movimentar o rotor em determinado sentido. Na *Tabela 7,* tem-se a sequência de chaveamento para rotação do motor no sentido horário visto de frente ao eixo.

Tabela 7 - Sequência chaveamento Motor no.: CO-BL23B46-02-02RO

Hall A	Hall B	Hall C	Fase A	Fase B	Fase C
0	1	0	Nível Baixo	Desativado	Nível Alto
1	1	0	Desativado	Nível Baixo	Nível Alto
1	0	0	Nível Alto	Nível Baixo	Desativado
1	0	1	Nível Alto	Desativado	Nível Baixo
0	0	1	Desativado	Nível Alto	Nível Baixo
0	1	1	Nível Baixo	Nível Alto	Desativado

Fonte: Adaptado de ALLTRONICS (2013)

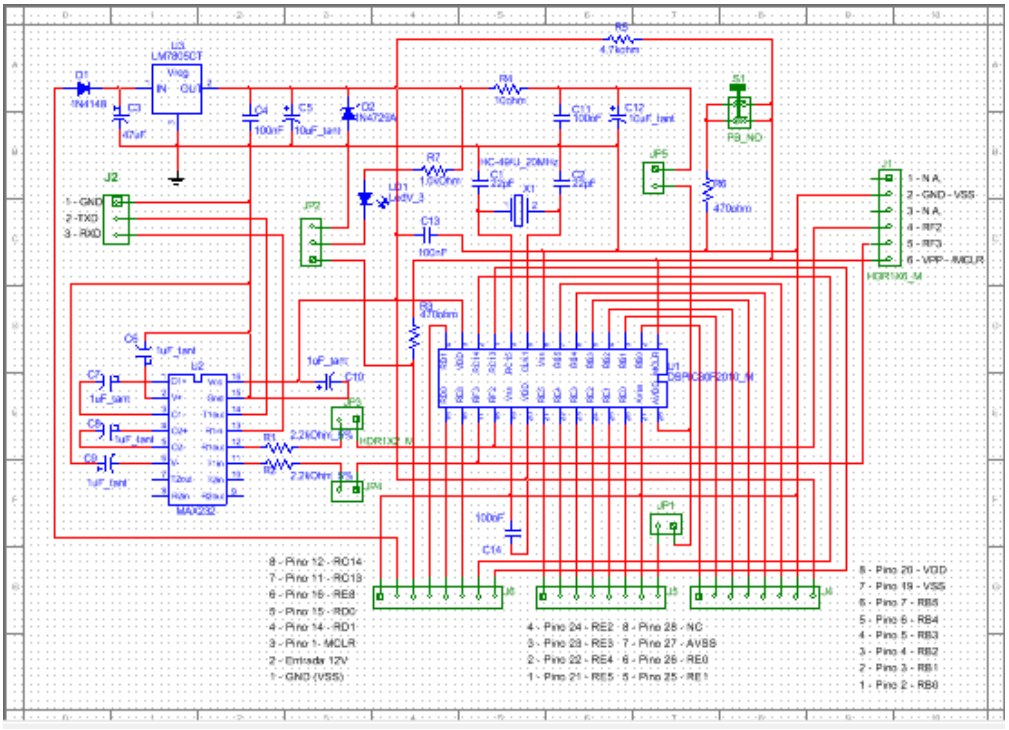
A sequência de chaveamento apresentada na *Tabela 7* é invertida caso o desejo seja girar o motor no outro sentido, e nota-se, ainda, que em nenhum momento há possibilidade para ativação simultânea das três fases, bem como leitura de três valores iguais no sensor hall.

3.1.2 **Placa de controle**

No desenvolvimento do hardware de controle necessário para operar o motor BLDC, optou-se pela utilização de uma plataforma de desenvolvimento dsPIC30F homologada, a fim de minimizar tempo de desenvolvimento, tornando o projeto modular e versátil, onde os pré-requisitos de funcionamento do microcontrolador dsPIC30F2010 se encontram funcionais, do ponto de vista de hardware, logo é necessário o desenvolvimento de um módulo de potência para utilização nessa topologia.

Na *Figura 29*, apresenta-se o esquemático da placa de controle utilizada.

Figura 29 - Esquemático placa de controle



Fonte: O autor (2017)

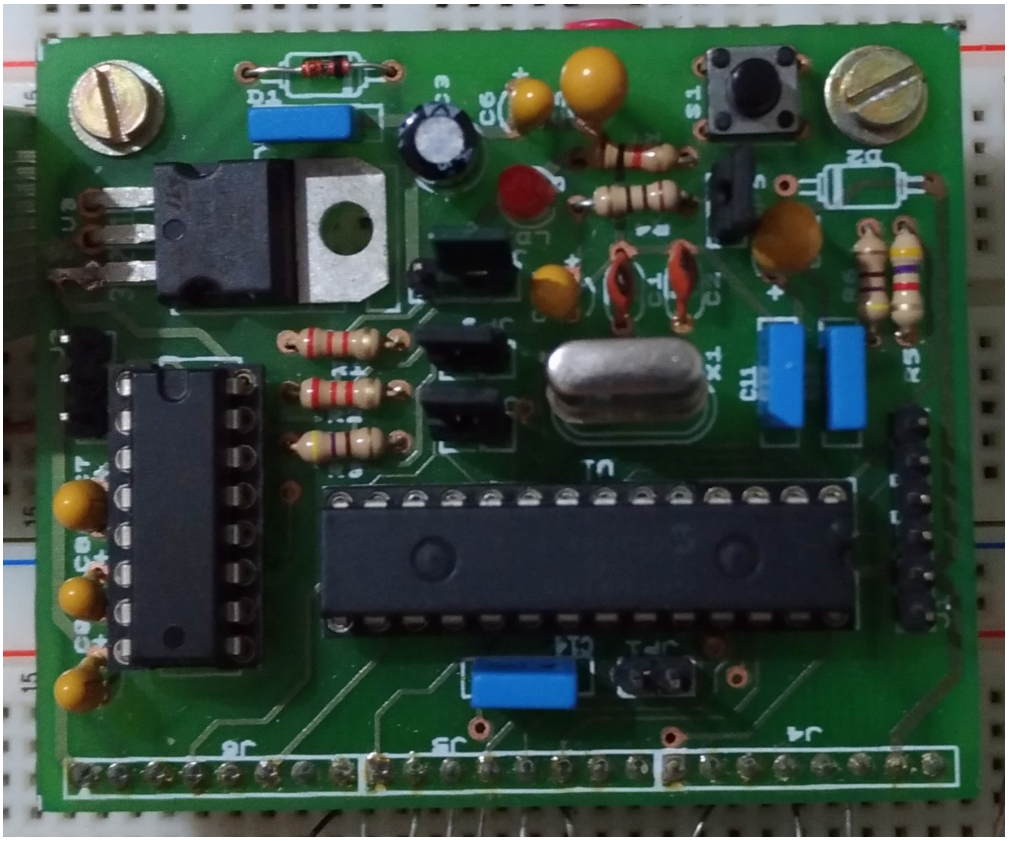
Na *Figura 29* pode-se, observar um circuito baseado no microcontrolador *dsPIC30F2010* que utiliza um cristal ressonante de 20 Mhz, regulador de tensão *LM7805*, LED indicador de condição multifunção, Circuito Integrado *MAX232* para utilização do protocolo serial, interface com pinos para gravação do firmware através do gravador *PICKIT3*, e interface em barramento com pinos que dão acesso aos *PORTs* do microcontrolador e às alimentações necessárias ao funcionamento.

Analisa-se que essa plataforma foi desenvolvida para utilização em múltiplos projetos, e atende a demanda para o desenvolvimento deste.

Na *Figura 30*, tem-se a vista superior da placa de controle montada em uma

matriz de contatos.

Figura 30 - Vista superior Placa de Controle na matriz de contatos



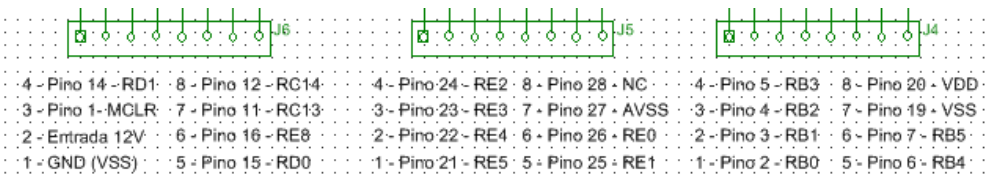
Fonte: O autor (2017)

Nota-se, na *Figura 30*, que o barramento de acesso aos *PORTs* do microcontrolador e suas alimentações encontram-se posicionados na face inferior da placa, o que permite o encaixe de maneira facilitada à matriz de contatos, ou a qualquer outro circuito projetado para esse fim.

3.1.2.1 Pinos Utilizados

Na *Figura 31*, tem-se a disposição do barramento.

Figura 31 - Barramento Placa de controle



Fonte: O autor (2017)

Identifica-se que o barramento apresentado na *Figura 31* é dividido em 3

interfaces, nomeados *J6*, *J5* e *J4*, onde cada posição do respectivo *J* é conectada a um pino do microcontrolador, descrita com o nome do pino e sua referente porta, ou apenas com uma referência de alimentação quando uma fonte de energia.

Para implementação do controle de um motor de imãs permanente com a utilização de sensores Hall e acelerador manual, tem-se condição prévia à utilização de seis pinos referentes aos PWMs de controle das fases do motor, três pinos referentes aos sensores de posicionamento Hall e um pino referente ao circuito acelerador. Adicionados recursos avançados ao nosso controle, utiliza-se mais cinco pinos, um para cada circuito adicional: Detecção de fonte de alimentação 24V; sensor de corrente; seletor de sentido de rotação; botão para partida / parada e LED indicador de status.

A Tabela 8, indica os pinos utilizados do microcontrolador.

Tabela 8 - Mapeamento das ligações do dsPIC30F2010

Pino	Função Pino	Ligação dos pinos aos módulos utilizados
1	MCLR	Reset e pino 6 para gravação in-circuit
2	EMUD3/AN0/VREF+/CN2/RB0	Detecção de fonte de alimentação 24V
3	EMUC3/AN1/VREF-/CN3/RB1	Sensor de corrente
4	AN2/SS1/CN4/RB2	Controle Velocidade
5	AN3/INDX/CN5/RB3	Sensor de posição do motor Hall_A
6	AN4/QEA/IC7/CN6/RB4	Sensor de posição do motor Hall_B
7	AN5/QEB/IC8/CN7/RB5	Sensor de posição do motor Hall_C
8	VSS	Ground
9	OSC1/CLK1	Cristal 20MHz para clock
10	OSC2/CLKO/RC15	Cristal 20MHz para clock
11	EMUD1/SOSC1/T2CK/U1ATX/CN1/RC13	Indicador de Atividade Led
12	EMUC1/SOSC0/T1CK/U1ARX/CN0/RC14	Botão de Partida e Parada
13	VDD	Fonte 5,00V
14	PREMUD2/OC2/IC2/INT2/RD1	Não utilizada
15	EMUCS/OC1/IC1/INT1/RD0	Não utilizada
16	FLTA/INT0/SCK1/OCFA/RE8	Circuito de sentido de rotação
17	PGD/EMUD/U1TX/SDO1/SDA/RF3	UART1 ligada ao Max 232
18	PGC/EMUC/U1RX/SDI1/SDA/RF2	UART1 ligada ao Max 232
19	VSS	Ground
20	VDD	Fonte 5,00V
21	PWM3H/RE5	PWM5
22	PWM3L/RE4	PWM4
23	PWM2H/RE3	PWM3
24	PWM2L/RE2	PWM2
25	PWM1H/RE1	PWM1
26	PWM1L/RE0	PWM0
27	AVSS	Ground virtual
28	AVDD	VCC virtual

Fonte: O autor (2017)

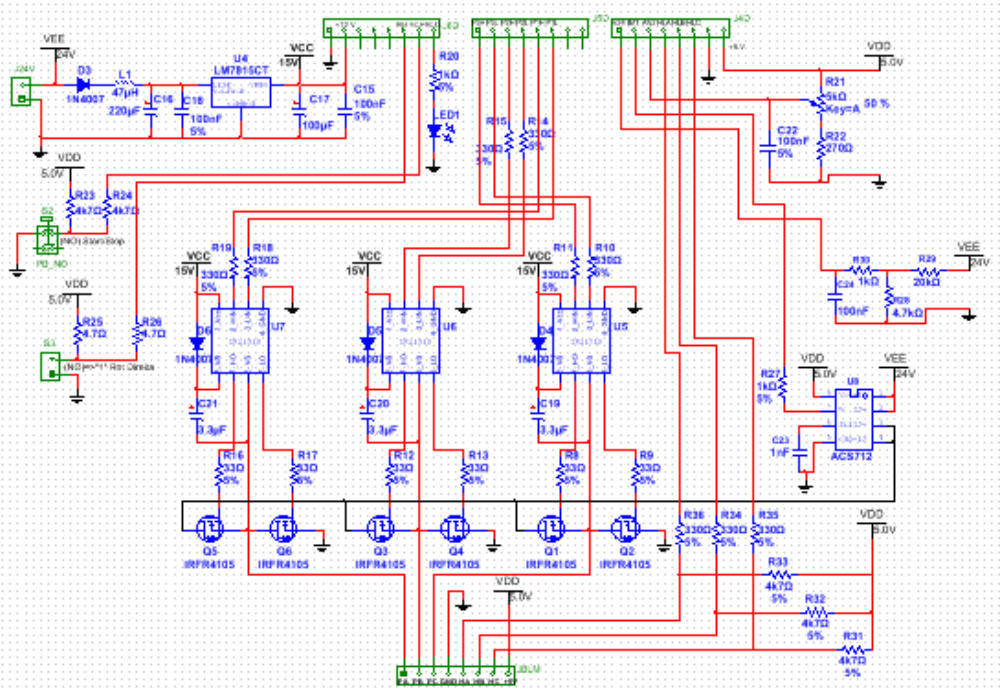
Observa-se na *Tabela 8* a disponibilidade de dois pinos sem utilização referente ao *PORTC* do microcontrolador, que poderá ser utilizado em implementações futuras.

3.1.3 **Placa de potência**

Baseado nas especificações do fabricante do microcontrolador, e utilizando-se do esquemático do ANEXO A como ponto de partida no desenvolvimento da

placa de potência, na *Figura 32* tem-se o esquemático do inicialmente desenvolvido para o módulo de potência do motor sem escovas.

Figura 32 - Esquemático módulo de potência



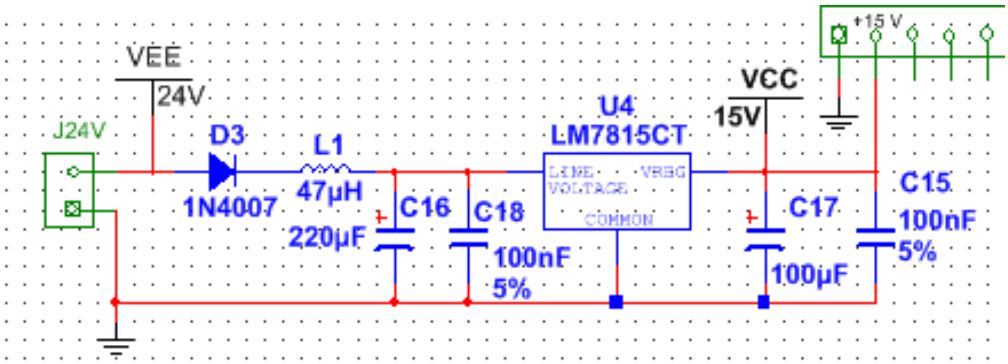
Fonte: O autor (2017)

3.1.3.1 Regulador de tensão

O regulador de tensão LM7815 tem função de rebaixar a tensão nominal da fonte principal de 24 V nominais para 15 V, a fim de suprir as necessidades do circuito da placa de controle, que possui outro regulador um LM7805, que diminui a tensão para 5 V, necessária pra o microcontrolador.

A *Figura 33* apresenta o esquemático do circuito utilizando o regulador LM7815

Figura 33 - Esquemático regulador LM7815



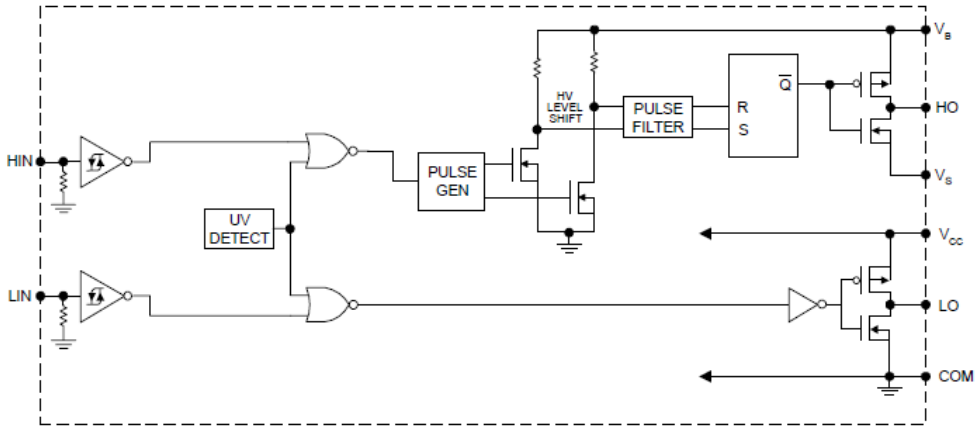
Fonte: O autor (2017)

Nota-se, na *Figura 33,* o circuito que compõe o regulador conta com um diodo na entrada para evitar que o usuário danifique o regulador ao ligar a fonte com polaridade equivocada e um filtro RLC passa-baixa, sugerido pelo fabricante, para minimizar possíveis efeitos de interferência causada pelo circuito possam impactar no desempenho do regulador; a tensão de 15 V do LM7815 também é necessária no circuito Driver High side e Low side proporcionado pelo circuito integrado IR2101S.

3.1.3.2 Conversor de Nível CC

Para acionar os transistores de potência, de maneira apropriada ao funcionamento do controle do motor é utilizado o driver IR2101S. De maneira geral, o driver é utilizado como um conversor CC-CC de um nível lógico TTL proveniente do microcontrolador para um nível suficiente de tensão e corrente capaz de acionar adequadamente os transistores de potência, que necessitam, por um curto período de tempo, uma potência significativa em seu gate que não é possível ser gerada diretamente pelas saídas do microcontrolador. Na *Figura 34* tem-se o diagrama de funcionamento do IR2101.

Figura 34 - Diagrama Funcionamento IR2101

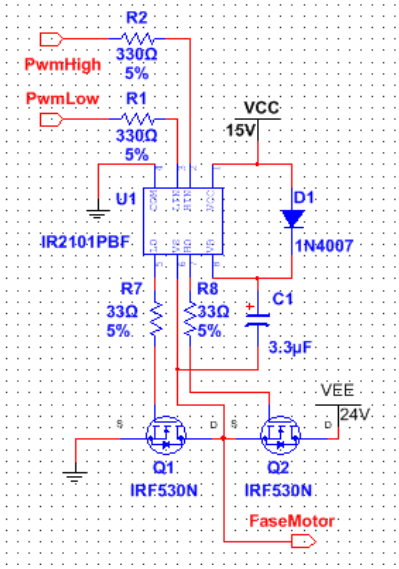


Fonte: Adaptado de Infineon Technologies (2004)

Nota-se, no diagrama da *Figura 34,* que o circuito é projetado para configurações complementares de operação em referência as saídas HO e LO.

Ainda, o circuito do IR2101 opera com topologia *bootstrap* através do diodo e capacitor, conforme a *Figura 35.*

Figura 35 - Esquemático high side e low side drive



Fonte: O autor (2017)

O efeito da topologia *bootstrap* cria na saída VS do IR2101 conectado entre o *Dreno* (d) do transistor do nível baixo e a *Fonte* (s) do transistor de nível alto um nível de tensão flutuante, através da carga e descarga do capacitor, o que permite o chaveamento de ambos os transistores em períodos alternados.

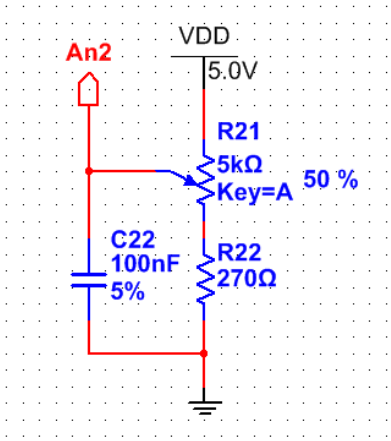
3.1.3.3 Transistor MOSFET de potência

O transistor MOSFET de potência IRF530N tem função de efetuar o chaveamento sobre as fases do motor. Cada fase do motor necessita dois transistores, cada um responsáveis por um estado, nível baixo ou nível alto. Os transistor de potência são utilizados em topologia Half Brigde, juntamente com o circuito IR2101, conforme nota-se na *Figura 35.*

3.1.3.4 Acelerador com potenciômetro

A capacidade de aceleração e desaceleração do motor enquanto em funcionamento se dá por um potenciômetro, utilizado conforme a topologia da *Figura 36*.

Figura 36 - Esquemático acelerador com potenciômetro



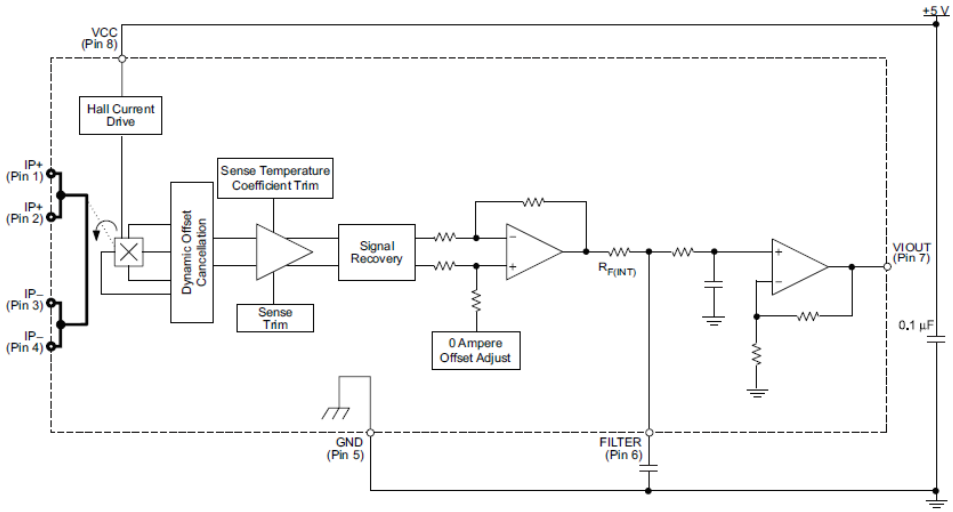
Fonte: O autor (2017)

Observa-se que o circuito apresentado na *Figura 36* têm o capacitor para reduzir os efeitos de interferência do circuito e um resistor a fim garantir uma resistência suficiente funcional mesmo com o potenciômetro em seu valor mínimo. Com isso, haverá um valor de tensão na porta do microcontrolador responsável pela aceleração, impedindo que o motor pare indevidamente por falta de aceleração, o que implica que mesmo sem aceleração nenhuma no potenciômetro, o motor continua a girar em uma velocidade reduzida, o que infere na necessidade de um comando especial para a parada do motor. Os valores nominais das resistências levam em conta as limitações de corrente e tensão informadas pelo fabricante do microcontrolador.

3.1.3.5 Sensor de Corrente

O circuito integrado ACS712 é utilizado como forma de sensoriamento da corrente consumida pelo circuito. Esse componente tem seu funcionamento através de efeito hall, onde a corrente é percebida pelo efeito de seu campo gerado. A *Figura 37* apresenta o diagrama do sensor ACS712-5B

Figura 37 - Diagrama Funcionamento ACS712



Fonte: Adaptado de Allegro MicroSystems (2017)

Nota-se, na *Figura 37,* as entradas IP+ e IP- formam um curto circuito no interior do circuito integrado, não havendo interferência de nenhum circuito interno na passagem da corrente.

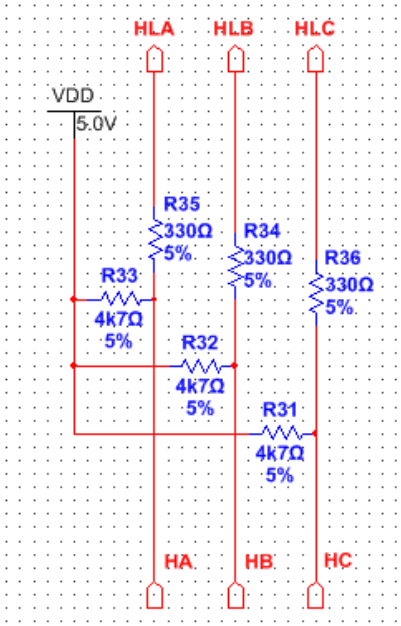
O sensor opera com uma corrente máxima de detecção de -5 A a 5 A, com uma tensão de referência de 5 V, sendo referenciada como 0 V quando -5 A, 2,5 V quando não a corrente, e 5 V quando tem-se 5 A. Em nosso caso, não esta prevista nenhum corrente rever, pois para girar o motor não depende do sentido da corrente, e sim do sentido em que as fases do motor são ativadas. Com isso, em nossa aplicação, o valor minimo de tensão esperado na porta do conversor AD do microcontrolador será de 2,5 V e o maior valor na corrente máxima de 5 A será de 5 V.

3.1.3.6 Pull up Encoder sensor hall do motor

Para utilização do sensores hall do *encoder* do motor, optou-se por uma topologia com resistor de *pull up* externo ao microcontrolador. Nos testes realizados com a utilização da opção de um *pull up* interno ao microcontrolador, não se têm os

resultados esperados. Na *Figura 38* tem-se o esquemático adotado.

Figura 38 - Esquemático pull-up para o sensor hall



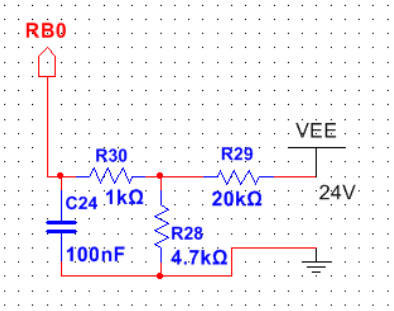
Fonte: O autor (2017)

Tem-se na *Figura 38* o resistor de 4,7 kΩ ligado à fonte de 5 V em uma das extremidades, um resistor de 330 Ω ligado ao microcontrolador e o sensor hall e na outra extremidade. Assim, quando o sensor hall tem seu valor de nível alto, cria um caminho para o terra tornado a porta do microcontrolador em nível baixo, limitada a corrente pelo resistor de 4,7 kΩ; quando o sensor hall tem seu valor de nível baixo, torna-se um circuito aberto e a porta do microcontrolador em nível alto, limitada à corrente pela série do resistor de 330 Ω e resistor de 4,7 kΩ. Os valores nominais das resistências levam em conta as limitações de corrente e tensão informadas pelo fabricante do microcontrolador.

3.1.3.7 Detector fonte

O detector de fonte foi adicionado ao projeto com intuito de verificar se há tensão de fonte. Com isso, se o valor da fonte for diferente ao esperado, o circuito não acionara o motor. Outra implementação a partir dessa leitura, o cálculo da potência aplicada ao circuito em tempo real. A *Figura 39* apresenta a topologia do circuito.

Figura 39 - Esquemático detector de tensão



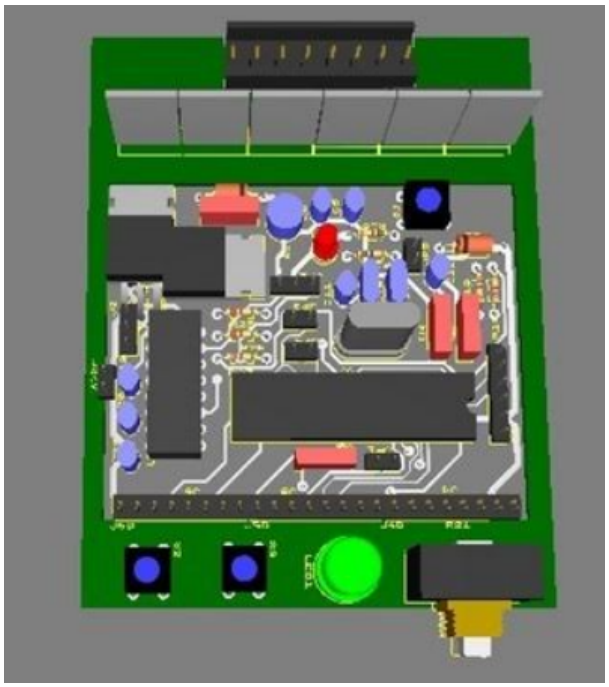
Fonte: O autor (2017)

No esquemático da *Figura 39* tem-se um divisor de tensão formando pelos resistores de 20 kΩ e 4,7 kΩ afim de reduzir a tensão aplicada a porta RB0 do microcontrolador e limitar sua corrente, e um filtro passa baixa formado pelo resistor de 1 kΩ e o capacitor de 100 nF com a finalidade de evitar leituras indevidas devido a possíveis interferências de alta frequência na fonte. Os valores nominais dos componentes utilizados atendem as limitações de corrente e tensão informadas pelo fabricante do microcontrolador.

3.1.3.8 Circuito na matriz de contatos

Em comparação com o desenvolvimento inicial, verificou-se uma discrepância nas disposição dos pinos referentes ao componente IR2101, e em um primeiro momento, desenvolveu-se uma modificação de layout, para tornar o circuito mais compacto, optou-se por uma disposição de sobreposição das placas, conforme o modelo 3D da *Figura 40.*

Figura 40 - Vista superior em 3D do layout com placas sobrepostas



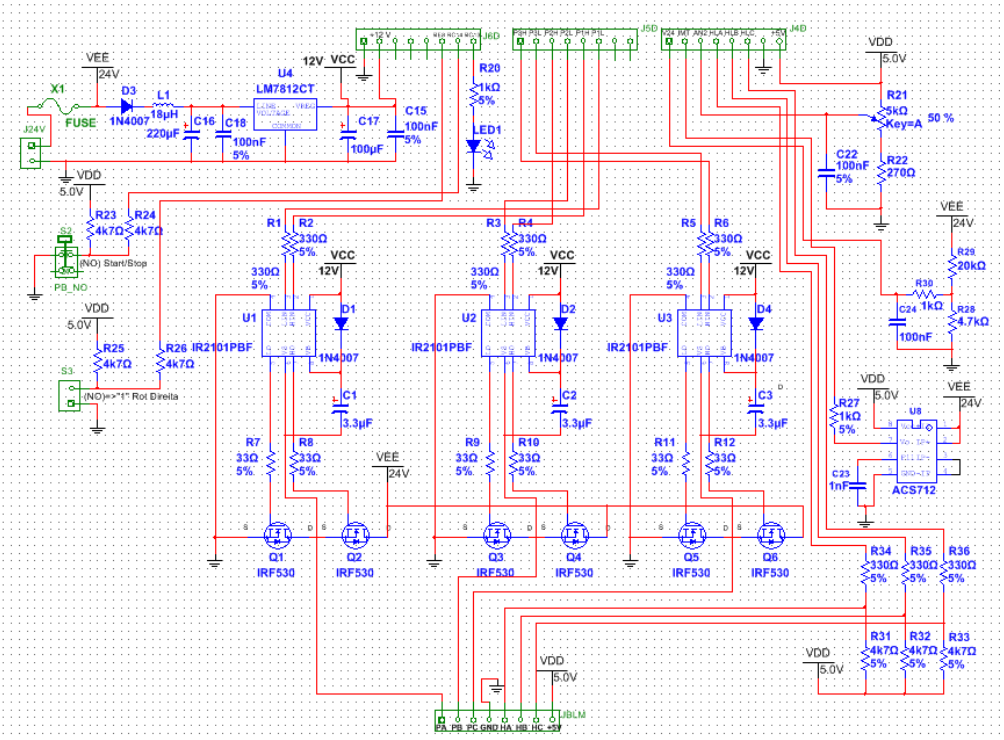
Fonte: O autor (2017)

Nota-se, na *Figura 40,* que a placa de controle (em cinza) tem disposição superior em relação a placa de potência (em verde).

Porém, na construção desse layout, tem-se dificuldade em efetuar medidas em pontos específicos do circuito criando uma série de desvantagens enquanto protótipo. Logo, efetua-se um novo designer sem a sobreposição

Na *Figura 41,* tem-se o novo esquemático desenvolvido para atender o novo layout e a nova disposição de pinos do componentes IR2101.

Figura 41 - Esquemático módulo de potência modificado

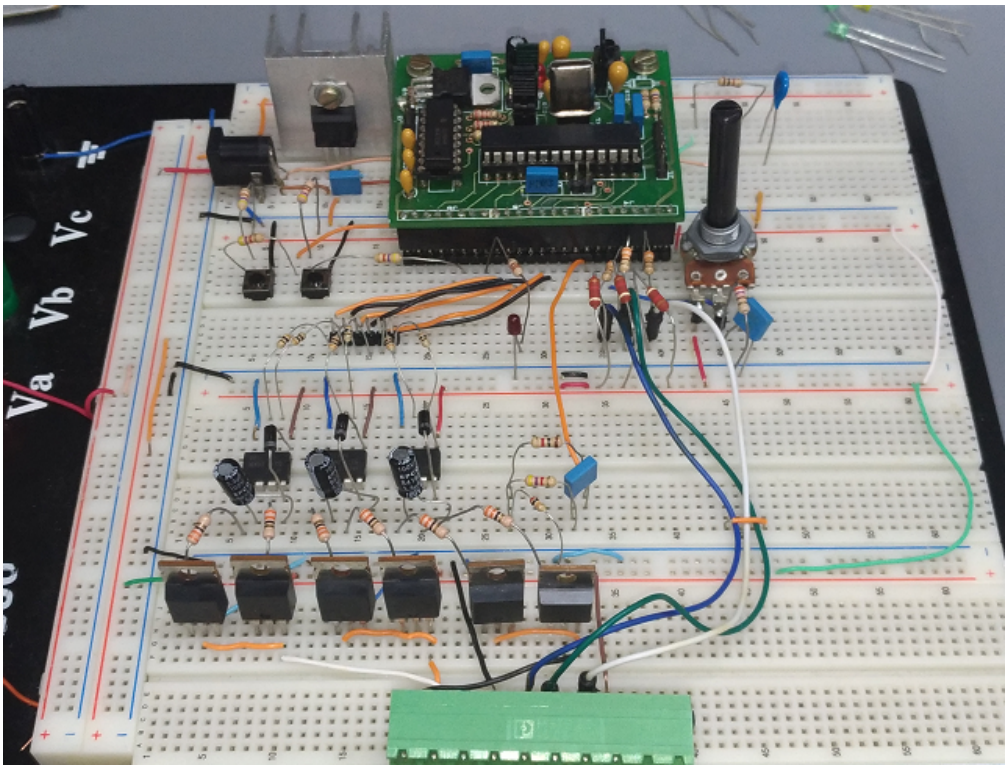


Fonte: O autor (2017)

Destaca-se que, na *Figura 41, a* adição de um fusível de proteção na entrada e a substituição do LM7815 por um LM7812, equivalente em funcionamento em nossa topologia, não causa mudanças comportamentais significativas.

Logo, com as definições da topologia do circuito definidas, na *Figura 42,* tem-se a montagem na *matriz de contatos*.

Figura 42 - Circuito montado na matriz de contatos



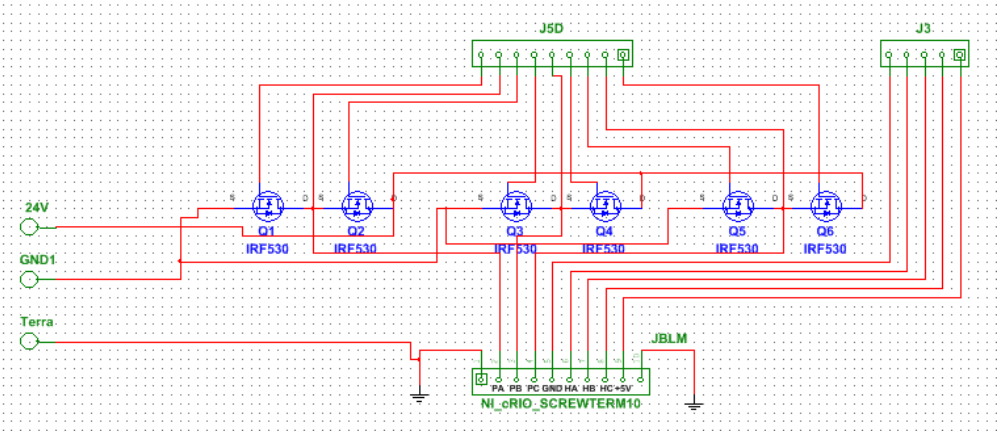
Fonte: O autor (2017)

Nota-se, na *Figura 42,* que um dissipador de calor foi acoplado no regulador de tensão LM7812 a fim de evitar problemas com possíveis aquecimentos; não implementa-se o fusível na *matriz de contatos* devido a utilização de fontes de bancada com controle de corrente; o mesmo será implementado apenas no protótipo em placa.

3.1.3.9 Circuito de potência em placa auxiliar

Conforme verifica-se no tópico de testes, muito ruído elétrico foi detectado em nosso circuito, tornando impossível realizar testes avançados. A partir dessa problemática que, em hipótese, pode ter causa devido às problemáticas de mal contato intrínsecas da matriz de contatos, e também à significativa corrente que o motor demanda para o dimensionamento das trilhas e fios usados na montagem da matriz de contatos, optou-se pelo desenvolvimento de um modulo de potência auxiliar conforme o esquemático da *Figura 43*.

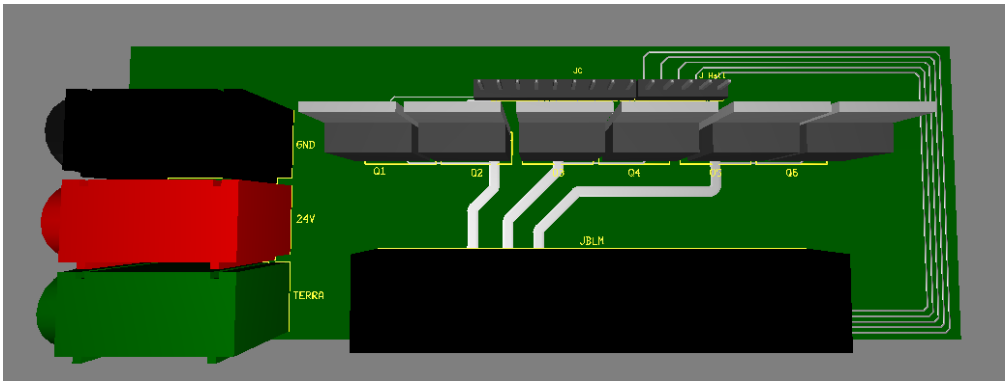
Figura 43 - Esquemático módulo de potência auxiliar



Fonte: O autor (2017)

Nota-se que, no esquemático da *Figura 43*, apenas os transistores de chaveamento são inseridos nessa topologia, pois são a demanda de maior corrente do circuito, a fim de promover um melhor desempenho tenta-se minimizar o ruído elétrico apresentados nas ligações da matriz de contatos. Na *Figura 44*, tem-se a vista superior do modelo em 3D do layout da placa auxiliar.

Figura 44 - Vista superior em 3D do layout da placa auxiliar



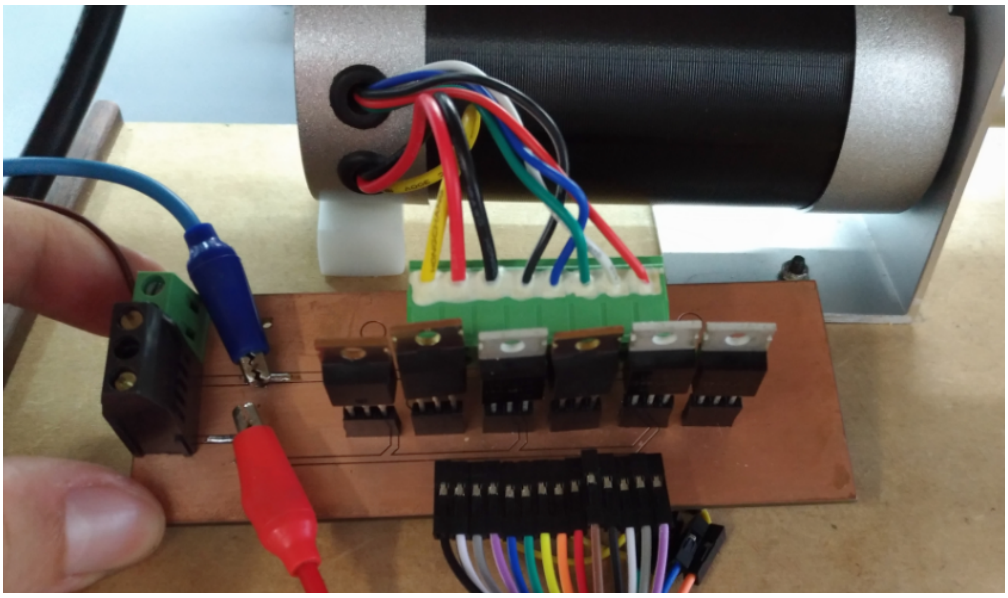
Fonte: O autor (2017)

Destaca-se, na *Figura 44,* as trilhas responsáveis pela alimentação do motor foram alargadas tolerando uma maior passagem de corrente elétrica. A largura é estimada em uma tabela de aproximação do fabricante da placa de circuito impresso, levando em conta uma corrente máxima de 4 A.

A significativa simplicidade dessa placa traz agilidade e confiabilidade no seu desenvolvimento, manufaturada em impressora de protótipo LPKF ProtoMat S63, disponível no laboratório de protótipos da Universidade Feevale.

Na *Figura 45,* apresenta-se a placa desenvolvida conectada ao motor.

Figura 45 - Placa auxiliar conectada ao motor



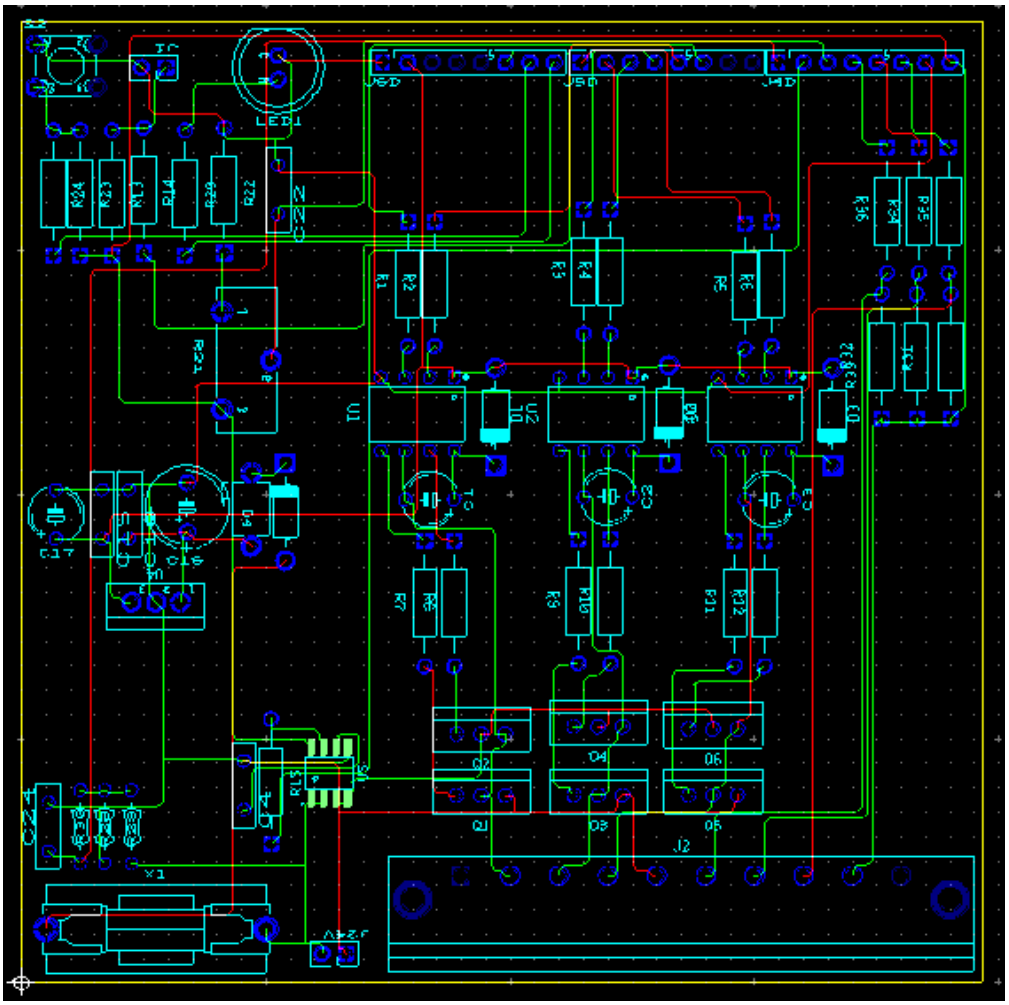
Fonte: O autor (2017)

Nota-se, na *Figura 45,* que os transistores se conectam à placa através de soquetes, o que facilitam a sua troca caso, por qualquer razão, algum deles esteja defeituoso; para a malhar de aterramento de ambas as faces projetou-se um conector para um terra extra, afim de verificar a sua influência no circuito; utiliza-se um cabo *flat* para ligação das conexões ao circuito da *matriz de contatos*; o conector utilizado na montagem, difere do projetado. Com isso foi possível aproveitar a furação para implementar pinos extras e efetuar a alimentação a placa diretamente com conectores jacaré, da fonte de bancada.

3.1.3.10 Placa protótipo final

Após os testes de funcionalidade e comprovações necessárias dos protótipos em matriz de contatos e da placa auxiliar desenvolvida, tem-se na *Figura 46* o diagrama da disposição e roteamento dos componentes da placa de potência.

Figura 46 - Diagrama de roteamento dos componentes placa de potência

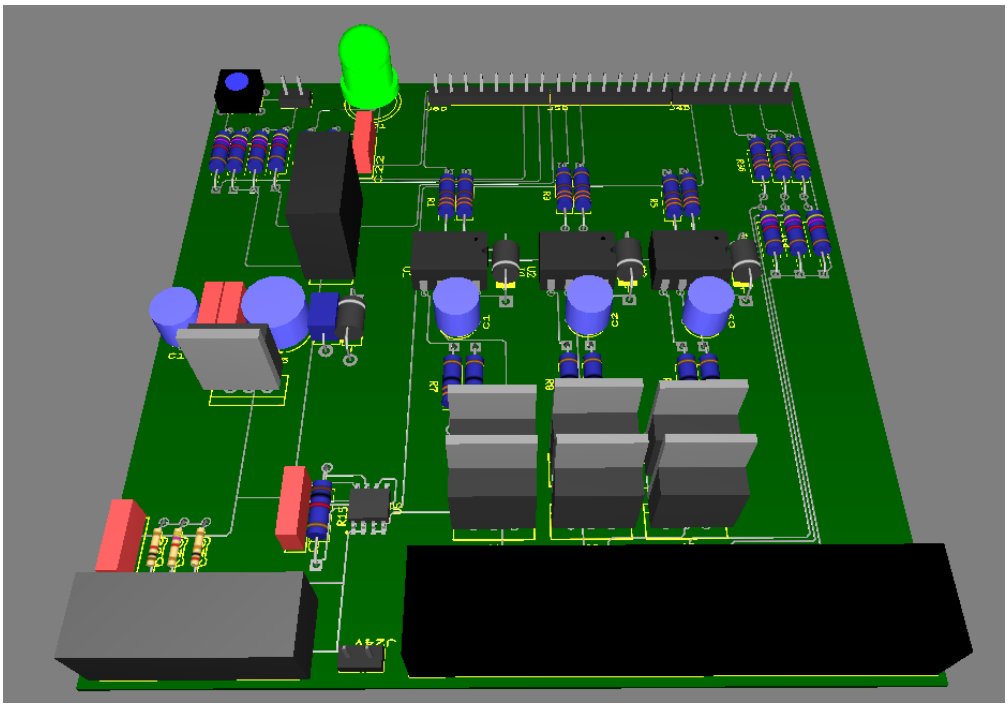


Fonte: O autor (2017)

Destaca-se, na disposição apresentada na *Figura 46*, uma área total de 100 cm^2, com os componentes distribuídos tendo em vista o agrupamento dos circuitos funcionais de maneira individualizada, como uma maneira intuitiva na análise de possíveis erros e facilitar as medidas na placa, em vista que trata-se de um protótipo.

A *Figura 47* apresenta a vista frontal em 3D do protótipo.

Figura 47 - Vista superior em 3D do layout

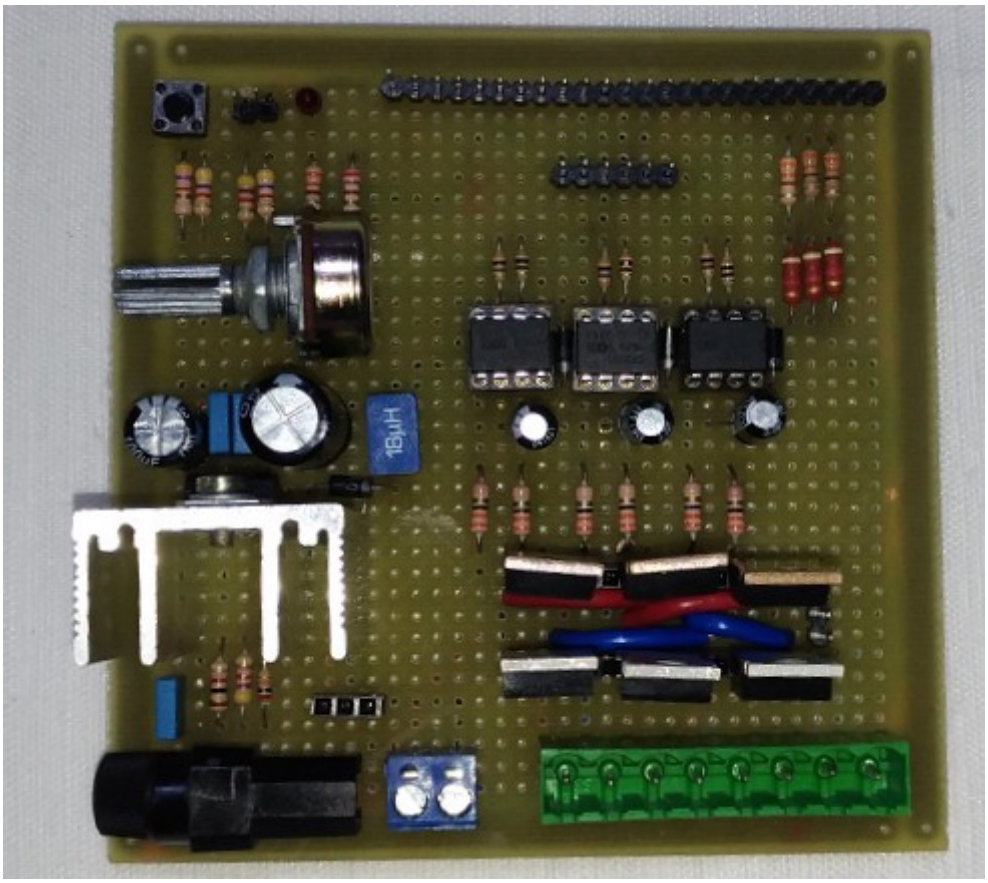


Fonte: O autor (2017)

Nota-se, na *Figura 47,* um arranjo intuitivo, com espaçamentos suficientes para medidas com osciloscópio, e implementações futuras, o que comprova a viabilidade de sua construção.

Na *Figura 48,* tem-se a visão superior da placa construída.

Figura 48 - Placa de potência



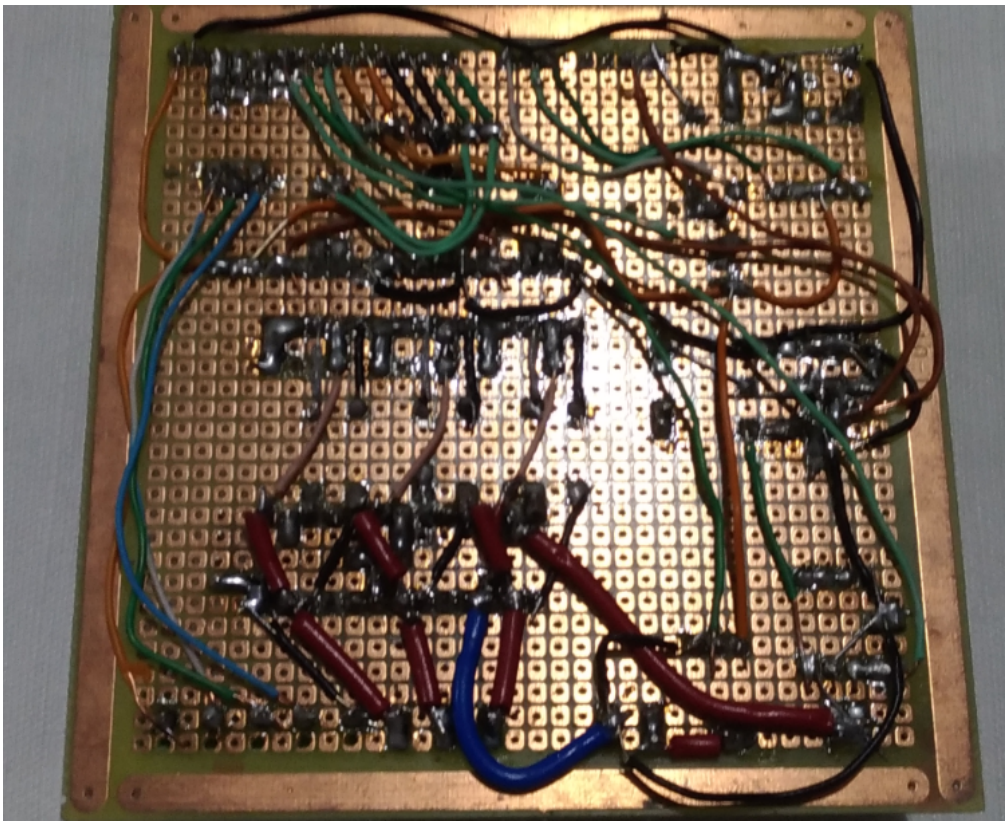
Fonte: O autor (2017)

Nota-se, na *Figura 48,* que para o seu desenvolvimento foi utilizada uma placa perfurada universal de 100mm^2 com a disposição dos componentes seguindo os padrões do desenvolvimento do modelo em *3D*; adicionou-se um dissipador de alumínio junto ao regulador de tensão e pinos de teste nas portas de controle do *PWM*, para leitura com osciloscópio junto ao microcontrolador.

Optou-se pela utilização de um módulo de sensor de corrente comercial com o sensor de corrente ACS712 amplamente utilizado na plataforma Arduíno, o que nos proporciona a sua utilização, mesmo tendo seu encapsulamento apenas em *SMD*; para isso, tem-se conector para o seu encaixe.

A *Figura 49* apresenta a vista inferior da placa, onde destaca-se a solda dos condutores aos pinos dos componentes.

Figura 49 - Vista inferior placa driver de potência

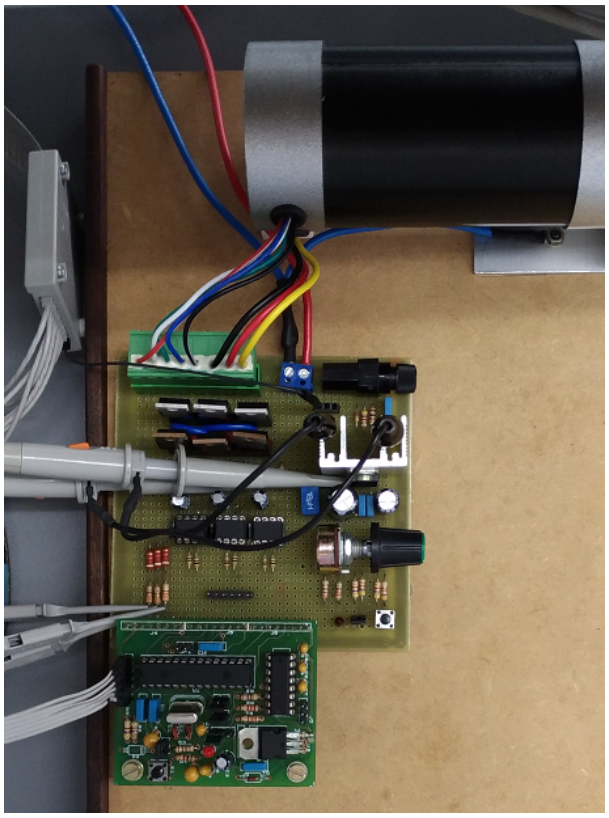


Fonte: O autor (2017)

Observa-se na, *Figura 49,* a utilização de condutores de maior capacidade de corrente entre os terminais do motor e os transistores de chaveamento.

Na *Figura 50,* tem-se a placa de potência acoplada ao motor e a placa de controle, compondo a banca de teste utilizada para o desenvolvimento.

Figura 50 - Disposição e acoplamento do hardware



Fonte: O autor (2017)

Nota-se que, na *Figura 50* a disposição dos componentes da placa proporcionam agilidade e organização na execução dos testes e implementações do software.

3.2 SOFTWARE DE CONTROLE

3.2.1 **Entradas digitais de ativação**

Para o processo de ativação do motor, foram determinados parâmetros como pré-requisitos para o motor iniciar.

Na *Figura 51,* tem-se o código responsável por implementar esse pré-requisitos.

Figura 51 - Código fonte da rotina principal

```
while(1){
while(S2)                               //botão não pressionado
{
    if( RunMotor == 1 && IM > 0x3FF)// Se corrente maior que 5A
    {
        Stop();                         //para o motor e
        for (i=0; i<10; i++)            // Pisca o LED 10 vezes.
            {
                LED1=!LED1;
                delayms(100);
            }
    }
}
while(!S2)                              //mudança de estado botão
{
    if( RunMotor == 1)                  // se o motor rodando
        Stop();                         // desativa o motor
    else                                // se motor parado
    {
        if (V24 > 511 && V24 < 1020)//Se tensão entre 12V e 24V
            Start();                    //ativa o motor
        else                            //se não
        {
            for (i=0; i<5; i++)         // Pisca o LED 5 vezes.
            {
                LED1=!LED1;
                delayms(100);
            }
        }
    }
}
        }
```

Fonte: O autor (2017)

Nota-se que, na *Figura 51,* para a liberação da inicialização do motor, são verificados parâmetros de tensão de alimentação e a corrente, através dos parâmetros *V24* e *IM.*

A verificação de tensão é feita após o botão de partida ser pressionado, e a verificação de corrente durante o giro do motor. Se algumas dessas condições não forem atendida, o *LED1* se comporta diferente para cada condição, tornando visualmente identificável.

A *Tabela 9* apresenta as condições do *LED1* caso algum evento previsto ocorra.

Tabela 9 - Tabela de identificação de condição LED.

Condição	Causa
Ligado	Motor em funcionamento
Desligado	Circuito sem alimentação
Piscando a cada 500ms	Aguardando pressionar botão de partida
Ao pressionar o botão, pisca 5 vez	Tensão de entrada fora dos parametros
Motor desligou sozinho e Led pisca 10	Corrente acima dos parâmetros.

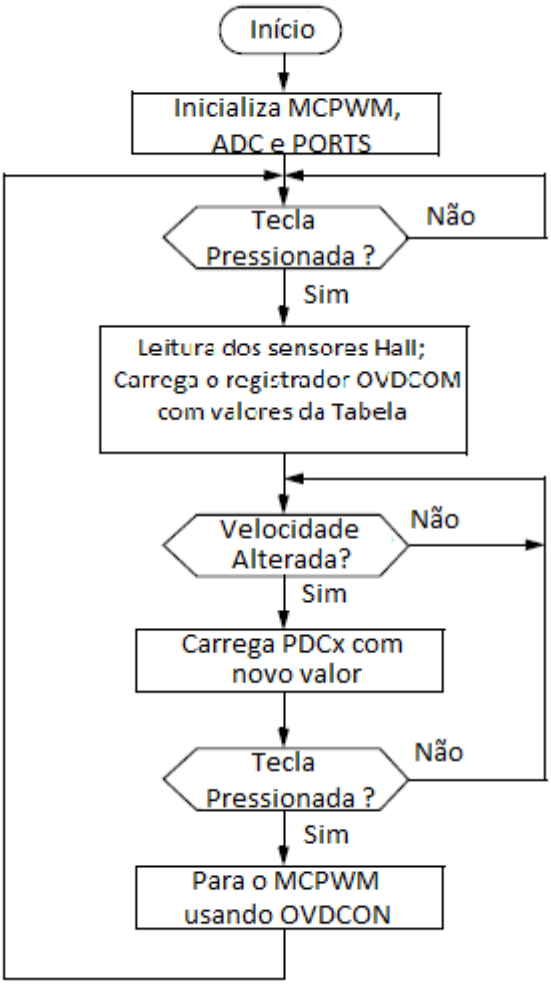
Fonte: O autor (2017)

Nota-se, através da *Tabela 9,* que apenas algumas condições pontuais têm abordagem, mas é possível a implementação de mais condições caso o projeto julgar necessário.

3.2.2 **Motor em funcionamento**

Na *Figura 52,* tem-se o fluxograma que implementa o funcionamento das rotinas.

Figura 52 - Fluxograma do motor em funcionamento



Fonte: O autor (2017)

Nota-se, pelo fluxograma da *Figura 52,* que o próprio giro do motor causa a interrupção responsável pela alteração das variáveis de controle, e a variável de velocidade define qual o número de vezes essa interação deve acontecer.

No Conversor ADC de 10 bits, constam os principais parâmetros de configuração no trecho de código que segue na *Figura 53*.

Figura 53 - Código inicialização do ADC

```
void InitADC10(void)
{
ADPCFG = 0xFFF8;     // PORTB = Digital exceção de RB0, RB1 e RB2 = analógico
ADCON1 = 0x0064;     // Ativa o MCPWM nas saidas PWM 1100100
ADCON2 = 0x0200;     // Amostragem dos 3 canais do Hall Simultaneamente
ADCON3 = 0x0080;     // Utilização do clock Interno RC
ADCHS = 0x0002;      // Conecta RB2/AN2 no CH0
IFS0bits.ADIF = 0;   // Limpa a flag de interrupção do AD
IEC0bits.ADIE = 1;   // Habilita a interrupção do AD
ADCON1bits.ADON = 1;// Liga o AD.
}
```

Fonte: O autor (2017)

Nota-se, na *Figura 53,* que a configuração da amostragem dos canais do sensor hall ocorrem simultaneamente, e o potenciômetro é definido como controle de velocidade.

Na leitura do sensor hall, adquiri-se a posição atual do motor. Esses valores, que possuem 6 combinações binarias possíveis, de 001 até 110, ou seja, de um até seis em decimal, correspondem as posições no vetor, que formam uma tabela de códigos de ação utilizados no registrador OVDCOM, responsável pela sobreposição do estado das saídas de largura de pulso que rotacionam o motor. Tem-se a tabela, em forma de vetor, para o giro para direita e para esquerda no trecho de código que segue na *Figura 54.*

Figura 54 - Trecho de código com vetores de sentido da rotação

```
//Sentido de rotação esquerda
unsigned int StateLoTable[] =
{0x0000, 0x0210, 0x2004, 0x0204, 0x0801, 0x0810, 0x2001, 0x0000};

//Sentido de rotação direita
unsigned int StateRoTable[] =
{0x0000, 0x2001, 0x0810, 0x0801, 0x0204, 0x2004, 0x0210, 0x0000};
```

Fonte: O autor (2017)

Tem-se que os valores do vetor são definidos segundo o manual da família do microcontrolador, e seu sequenciamento depende do sentido de rotação que o motor adotar.

No trecho de código da *Figura 55*, tem-se a implementação da rotina de start, responsável por retirar o motor do repouso.

Figura 55 - Trecho de código para iniciar o motor

```
void Start(void)
{
while(S2)                       //Aguarda liberação do botão
{
    HallValue = PORTB & 0x0038; // Mascara de bits em RB3,4 & 5
    HallValue = HallValue >> 3; // Deslocamento de bits 3 vezes
    Direction = S3;             // Direção de rotação
    if (Direction == 1)         // Se Direção 1 Rotação para direita
    OVDCON = StateRoTable[HallValue];// Carrega o registrador de sobreposição
    else                        // Se direção 0, Rotação para esquerda
    OVDCON = StateLoTable[HallValue];// Carrega o registrador de sobreposição
    PDC1 = 0x500;               // inicia com PWM de valor elevador
    PDC2 = PDC1;                // devido a inercia.
    PDC3 = PDC1;
    PWMCON1 = 0x0777;           //Habilita as saidas PWM
    delayms(10);                //Aguarda 10ms
    LED1 = 1;                   //Ativa o LED1
    RunMotor = 1;               //Altera flag para motor rodando
    goto fim;                   //Sai do laço While.
}
fim:;
}
```

Fonte: O autor (2017)

Nota-se que no trecho de código da *Figura 55*, a leitura dos sensores hall e do sentido da direção, com essas informações, o registrador de sobreposição tem seu valor definido e um valor de largura de pulso é definido igualmente nas saídas do PWM, que são ativadas, onde inicia-se o giro do motor. Após 10 ms o LED1 liga, e a flag de motor rodando é acionada.

Na *Figura 56,* tem-se a implementação da rotina *stop*, responsável por parar o motor.

Figura 56 - Trecho de código de parada do motor

```
void Stop(void)
    {
    while(S2)           //Aguarda liberação do botão
{
    PWMCON1 = 0x0700;   //Desabilita as saidas PWM
    OVDCON = 0x0000;    //Desabilita o Registrador de sobreposição
    delayms(10);        //Aguarda 10ms
    RunMotor = 0;       //Altera flag para motor rodando
    LED1 = 0;           //Ativa o LED1
    goto fim;           //Sai do laço While.
}
fim: ;
    }
```

Fonte: O autor (2017)

No código da *Figura 56*, tem-se a desativação das saídas PWM do registrador de sobreposição, logo a flag de condição do motor rodando e o LED1 são

desativados.

3.2.3 **Leitura e escrita via UART**

Tem-se os trechos de código de recepção e envio de dados via UART, ao final do APÊNDICE B, identificados como *Rotina de tratamento da Interrupção do recebimento de dados e Rotina para enviar o pacote de dados via UART, respectivamente.*

Tem-se, na rotina UART, a possibilidade do usuário de envio de comandos de alteração de inicialização e parada do motor. A rotina de recepção do microcontrolador constitui-se de uma leitura de quatro *bytes*, onde o primeiro *byte* da sequência deve conter o número de série que autoriza a implementação do comando, evitando que comandos indesejados causados por ruídos elétricos sejam implementados via UART. Os próximos dois *bytes* foram reservados para futuras implementações de controle, e o último *byte*, executa o comando referente a parada ou inicialização do motor.

Tem-se, na rotina UART, a possibilidade do usuário de recepção de dados referente ao acontecimentos no microcontrolador, como o estado de movimento motor. A rotina de envio do microcontrolador constitui-se de maneira semelhante a de recepção, com o envio de quatro *bytes*, onde o primeiro *byte* da sequência deve conter o número de série que autentica a sequência de *bytes que contém a* informação, evitando assim, comandos indesejados causados por ruídos elétricos sejam recebidos via UART. Os próximos dois *bytes* foram reservados para futuras implementações de controle, e o último *byte* determina o status referente a rotação ou não do motor.

3.3 TESTES E VALIDAÇÕES

3.3.1 **Teste com software interrupção de timer**

A fim de verificar a correta alocação do registrador OVDCOM e testes iniciais de hardware com a rotina do *APÊNDICE A*, optou por uma rotina de teste independente das leituras do sensor hall, onde a velocidade de rotação atende às interrupções do *timer*.

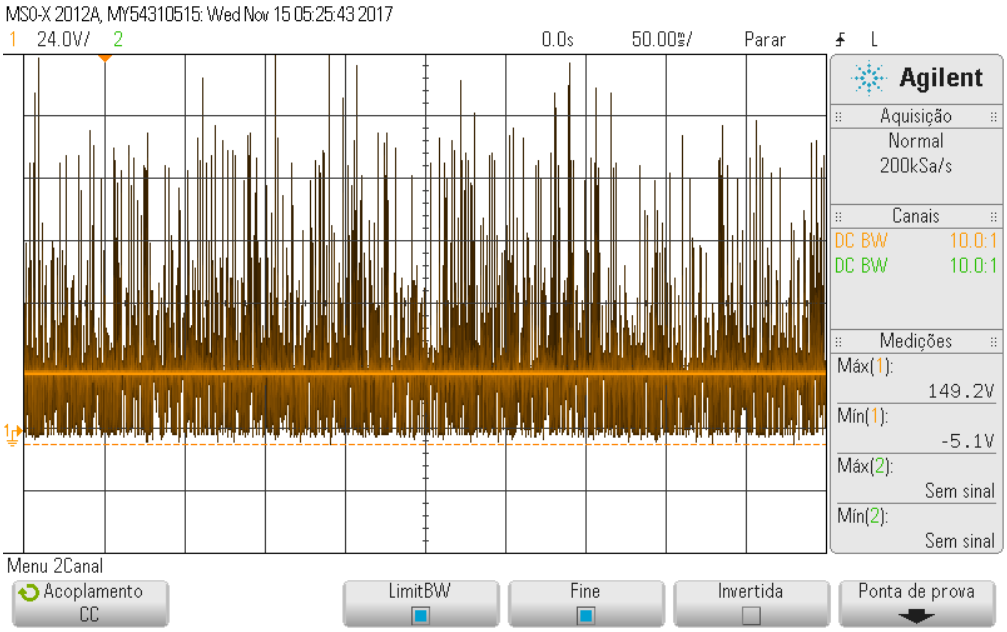
Nos testes iniciais, o motor gira em rotação constante conforme a interrupção do *timer*, e o ajuste da largura de pulso, através do potenciômetro, torna o

comportamento do giro suave, ao atingir um valor ideal; se abaixo o motor não gira; se acima, o motor entra em estado trepidante, pois o controle se dá temporalmente, e devido ao excesso de potência aplicada pela largura de pulso, ele passa do ponto ideal onde o registrador *OVDCOM* o espera.

Com o motor em funcionamento, e o circuito ainda na matriz de contatos, percebe-se um significativo aumento na temperatura do regulador de tensão LM7812, mesmo com os níveis de corrente significativamente abaixo dos parâmetros operacionais do componente e, após alguns minutos, o regulador interrompe seu funcionamento.

A *Figura 57*, mostra uma imagem do osciloscópio, onde a tensão da fonte é verificada com o motor em funcionamento através da interrupção do *timer.*

Figura 57 - Interferência durante rotação do motor

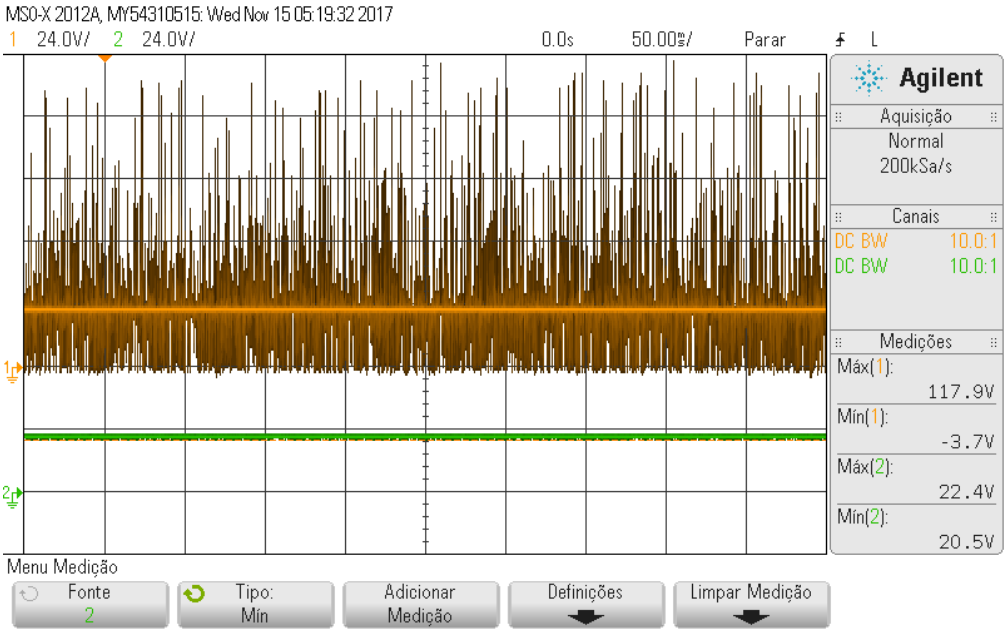


Fonte: O autor (2017)

Nota-se, na *Figura 57*, que com o motor em funcionamento surge uma significativa interferência de alta frequência sobre a fonte, com picos próximos à 150 V, que aumentam conforme a trepidação causada pelo excesso de largura de pulso, o que ultrapassa o valor nominal de entrada de tensão do regulador LM7812.

Para o contorno dessa problemática, o fabricante sugere em seu esquemático, que é usado como base nesse desenvolvimento, inserir um filtro passa-baixa na entrada do regulador. Na *Figura 58*, tem-se a comparação do sinal antes (Canal 1) e após o filtro (Canal 2).

Figura 58 - Comparação de sinais após aplicação do filtro



Fonte: O autor (2017)

Na *Figura 58*, percebe-se a efetividade do filtro, pois o efeito da interferência é reduzida, tornado o sinal tolerável pelo regulador.

Mesmo após a aplicação do filtro sobre o regulador, ainda percebe-se sobre a fonte do circuito forte nível de interferência, e na tentativa de diminui-lo, a placa auxilar de potência desenvolvida é utilizada, e conectada, promovem menor nível de mal contato dos componentes, e maior densidade de corrente aos transistores de chaveamento através de trilhas mais largas.

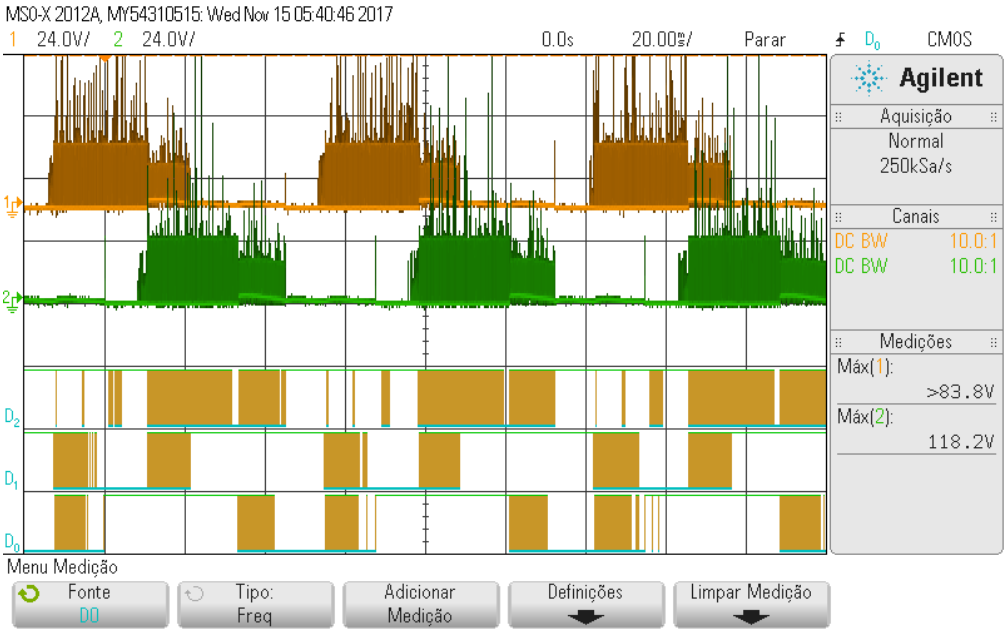
Porém, a placa auxiliar não tem o efeito esperado, apesar de uma sensível melhora no contexto geral, os níveis de interferência sobre a fonte continuam significativos, ainda assim não afetam o funcionamento do sistema nesse teste.

3.3.2 **Teste de malha aberta**

Através da rotina do *APÊNDICE B*, é efetuado os teste de malha aberta, onde através da leitura dos sensores hall, o potenciômetro define a velocidade do giro do motor.

Na *Figura 59* tem-se a Fase A do motor no Canal 1, e a Fase B no Canal 2, conectadas nas saídas analógicas do osciloscópio, e os canais digitais D0, D1 e D2 conectados aos sensores de efeito hall, diretamente as portas no microcontrolador.

Figura 59 - Análise dos sinal de fase do motor e sensor hall com interferência

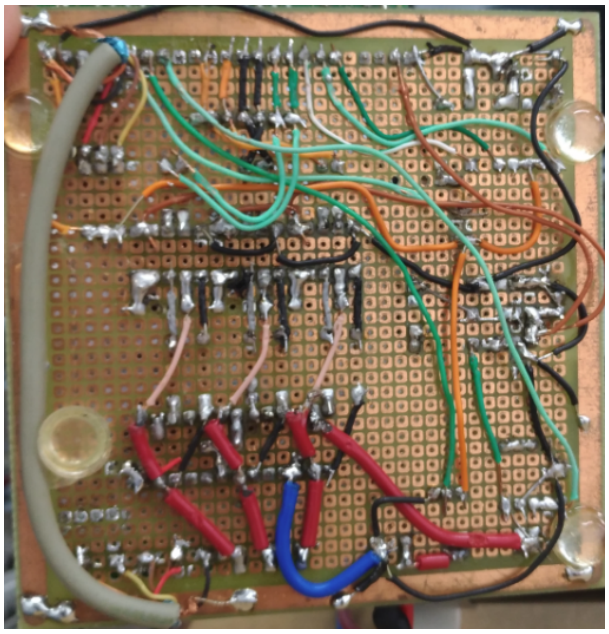


Fonte: O autor (2017)

Nota-se na *Figura 59*, que os canais analógicos contêm uma significativa interferência, assim como os canais do sensor hall, onde pulsos de tensão invalidam a leitura de forma clara dos sensores Hall, o que leva consequência desastrosas ao giro do motor por falsas leituras de condição e podem danificar as portas do processador.

Diante desse problema, na *Figura 60,* tem-se a substituição dos fios condutores do sensor hall da placa, por um condutor blindado, na tentativa da isolação elétrica desses cabos amenizar as interferências.

Figura 60 - Blindagem dos condutores do sensor hall

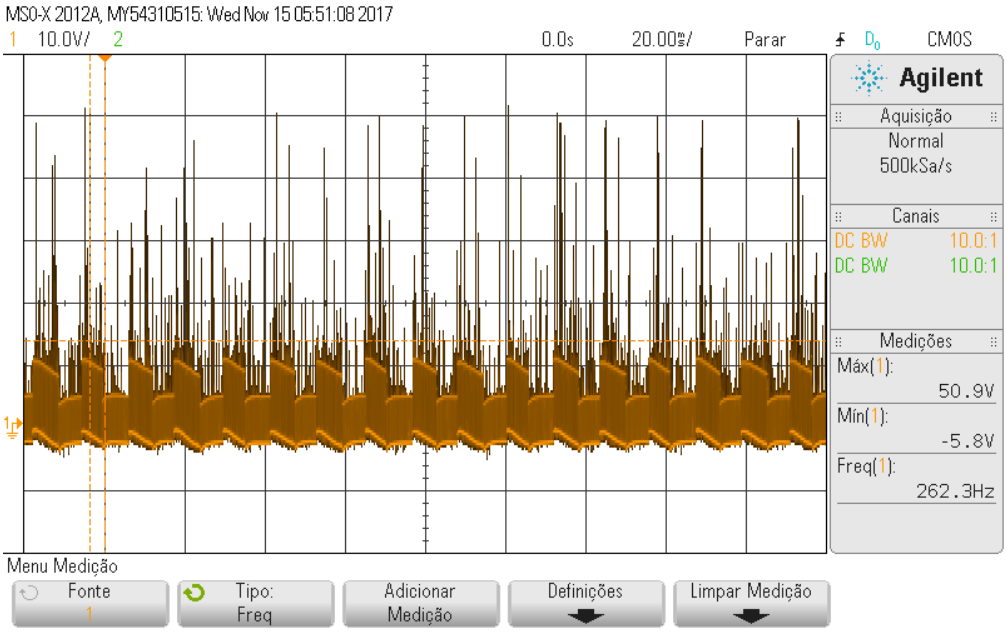


Fonte: O autor (2017)

Contudo, apesar de uma ligeira, melhora, o circuito ainda apresenta significativos níveis de ruídos elétricos, que impossibilitam uma leitura clara do sensor hall.

Em uma nova abordagem, efetua-se a medida da carcaça do motor em rotação, e conforme a *Figura 61*, encontra-se um significativo ruído elétrico.

Figura 61 - Interferência na carcaça do motor em funcionamento



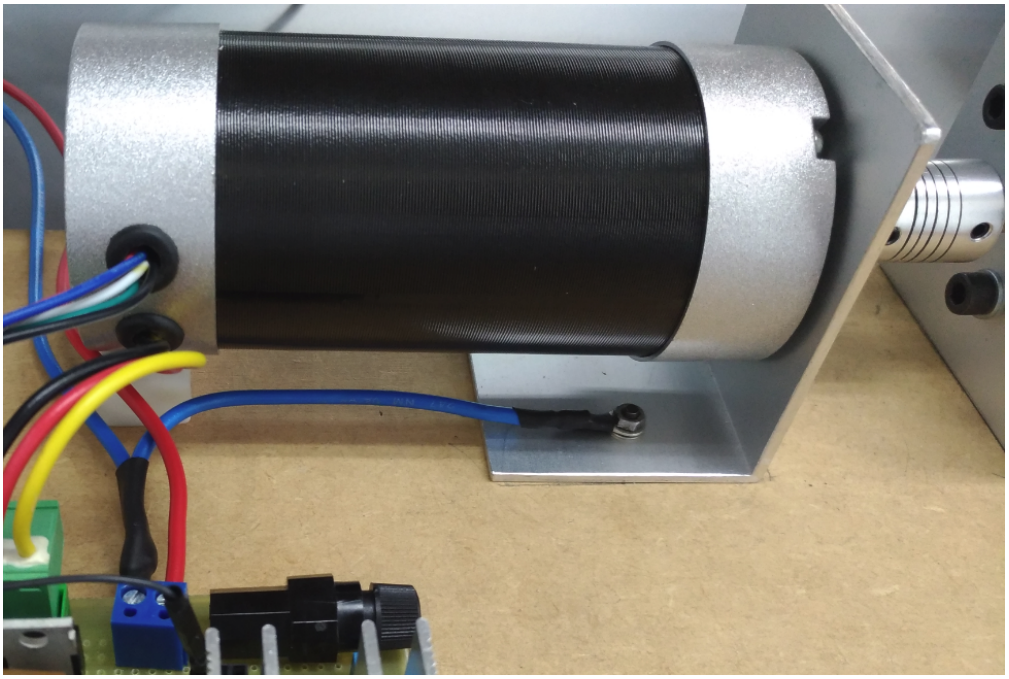
Fonte: O autor (2017)

Com essa constatação, atenta-se ao aterramento da carcaça do motor,

através de um conector especial, assim promove-se o balanceamento do terra entre a carcaça e a fonte de alimentação do circuito, com a utilização de condutores de maior espessura, voltado ao suprimento de maior capacidade de corrente exigidos pelo motor.

Na *Figura 62*, tem-se o conector ligado a carcaça do motor e ao terminal negativo da fonte de alimentação.

Figura 62 - Condutor especial para aterramento da carcaça do motor

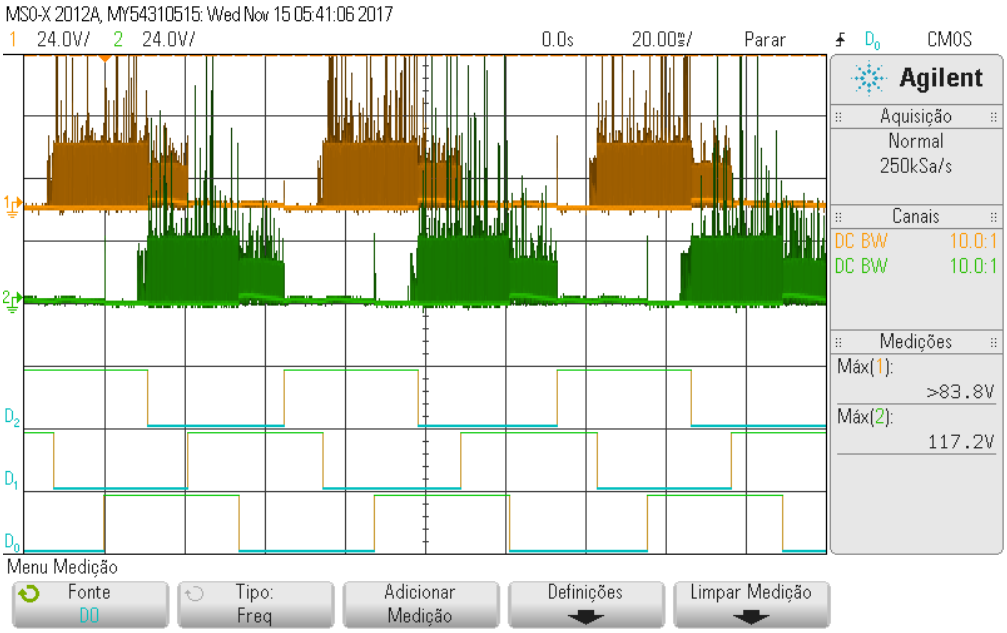


Fonte: O autor (2017)

Nota-se, na *Figura 62*, que o condutor utilizado tem seu comprimento minimo para diminuir as perdas e evitar o aumento da indutância.

Na *Figura 63* tem-se a nova leitura dos sensores hall após a implementação do aterramento da carcaça do motor

Figura 63 - Análise dos sinais do sensor hall sem interferência



Fonte: O autor (2017)

Nota-se na Figura 63 que os pulsos de tensão responsáveis pelas leituras errôneas não estão mais presentes nos canais digitais do osciloscópio, e o sinal fica adequado para utilização no microcontrolador.

Verifica-se ainda em testes de giro do motor, que ao ter seus sentido de rotação configurado para esquerda, ocorrem ruídos elétricos nas transições de rotação, o que não acontece quando configurado com sentido de rotação para direita.

Verifica-se que as configurações de calagem do motor estão setadas para giros em sentido de rotação direita. O motor utilizado nessa aplicação não atende uma transição de rotação suave em ambos os sentidos.

3.3.3 **Teste de malha fechada**

Através da rotina do APÊNDICE C, é efetuado os teste de malha fechada, onde, através da leitura dos sensores hall, o potenciômetro define o *offset* da velocidade do motor, e através da rotina de controle Proporcional e Integral mantem essa velocidade mesmo com a variação da carga do motor.

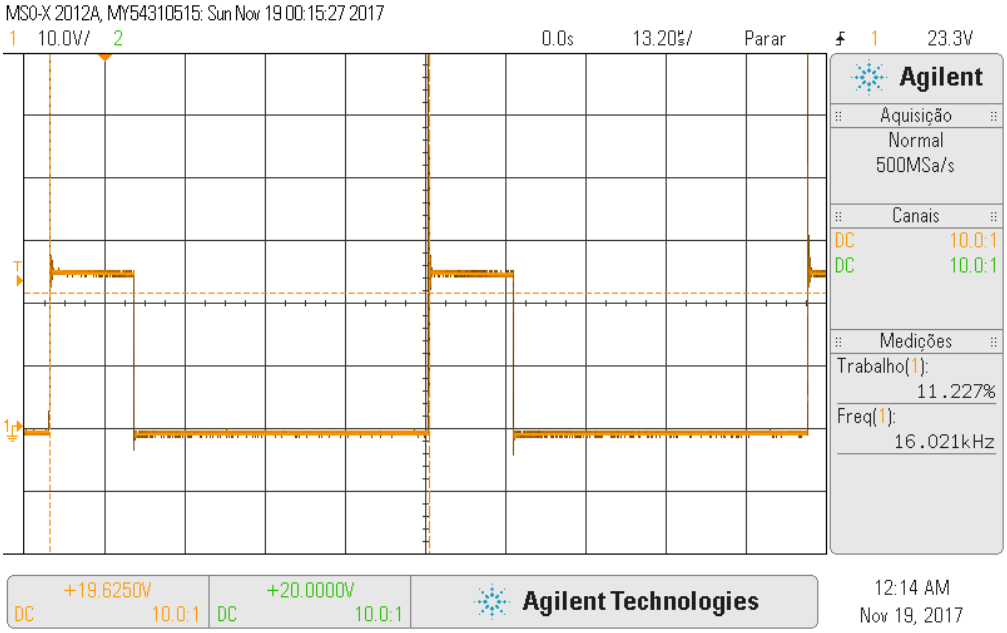
Para simular uma carga, um motor CC com potência similar, é acoplado ao eixo do motor brushless, a fim de verificar o comportamento do controle P.I.

Estima-se os parâmetros *Kp* e *Ki* do controle P.I. de maneira empírica, a fim de comprovar a sua ação em uma faixa de 900 RPM à 1200 RPM, arbitrada com intuito de testar a funcionalidade. Um ajuste fino dos parâmetro se faz necessária

para uma ação de maior eficiência e estabilidade ao longo das faixas de operação desse motor, devido à falta da disponibilidade da função de transferência pelo fabricante nas documentações oficiais disponíveis.

Na *Figura 64* tem-se a largura de pulso da Fase 1 do motor, com uma velocidade fixa através do potenciômetro, sem carga aplicada ao seu eixo.

Figura 64 - Largura de pulso sem carga no eixo e velocidade fixa

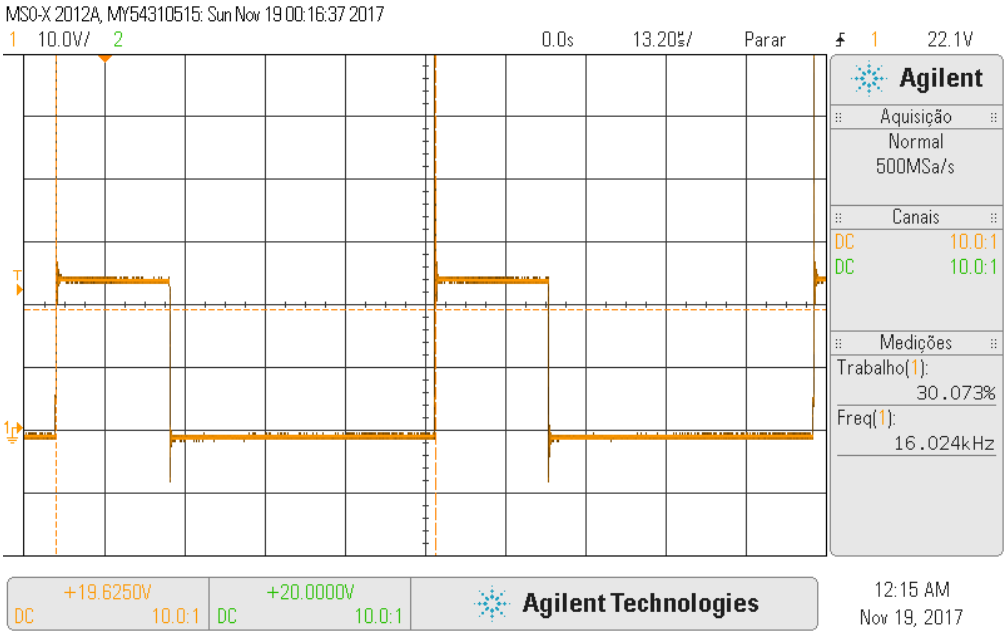


Fonte: O autor (2017)

Nota-se, na *Figura 64*, que a frequência de operação de 16 kHz do MCPWM, com um ciclo de trabalho de 11,22%. O motor opera com velocidade de 900 RPM tensão de 24 V e corrente de 300 mA.

Nos testes iniciais, uma carga é simulada diretamente com atrito no eixo, na *Figura 65 ,* tem-se a resposta a essa ação.

Figura 65 - Largura de pulso com carga no eixo

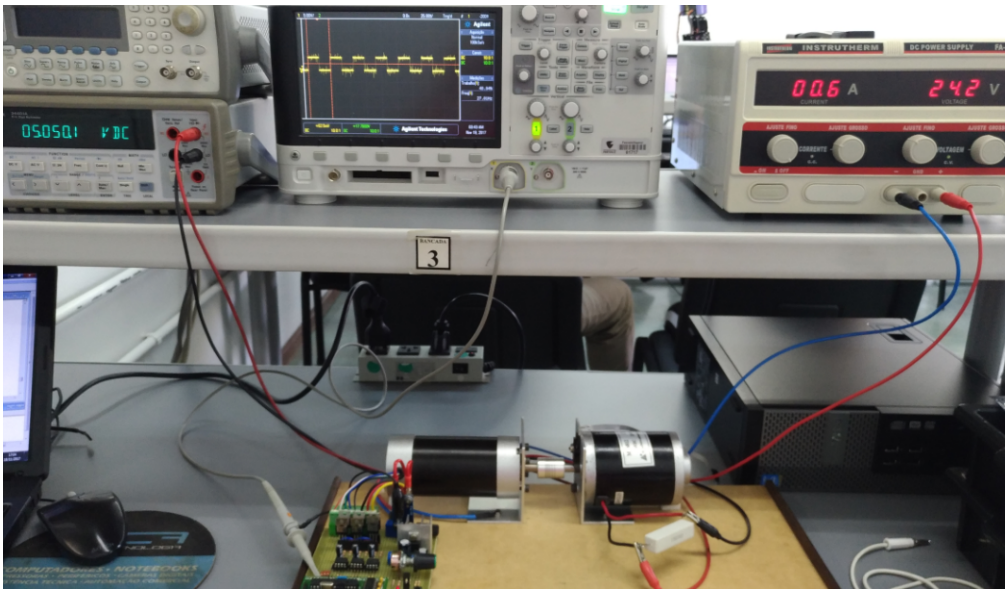


Fonte: O autor (2017)

Nota-se, na Figura 65, que o ciclo de trabalho se altera para 30% quando o atrito no eixo é efetuado, e a velocidade se mantém, com pequena variação. Porém a corrente de operação chega a 1,5 A, comprovando o correto funcionamento do controle P.I. aplicado.

Na *Figura 66* tem-se a bancada de teste de validação montada, utiliza-se o motor/gerador com um resistor de 10 Ω em seus terminais, acoplado ao eixo do motor, simula-se a carga.

Figura 66 - Bancada de teste de validação em malha fechada



Fonte: O autor (2017)

Apresenta-se na *Figura 66*, o osciloscópico ligado ao sensor hall para verificar a frequência de giro do motor, o multímetro ligado em paralelo ao terminais do resistor de carga no motor/gerador, para verificar a tensão gerada sobre o resistor e efetuar o cálculo da corrente; e a fonte de banca, configurada com tensão de saída de 24 V para alimentação do circuito e verificação da corrente consumida. Destaca-se que para realização do teste sem carga, o resistor de carga é retirado dos terminais do motor/gerador.

Na analise efetuadas, não estão sendo consideradas as diversas perdas mecânicas e térmicas, pois trata-se de uma avaliação exclusiva para verificar o funcionamento dos efeitos do controle P.I. ao ser inserida uma carga.

O teste é efetuado em duas rotações, para validar a faixa de frequência de operação do controle PI. Na *Tabela 10,* tem-se os valores obtidos no teste com rotação de 900 RPM, com tensão da fonte em 24 V.

Tabela 10 - Comparativo de carga com 900 RPM

Descrição	Medidas sem carga	Medidas com carga
Corrente da fonte	0,3 A	0,6 A
Potência da fonte	7,2 W	14,4 W
Tensão da carga	5,58 V	5,05 V

Fonte: O autor (2017)

Na *Tabela 10,* analisa-se a influência da carga de 10 Ω quando conectada ao

circuito, que demanda maior potência da fonte para manter os 900 RPM, sendo necessário o dobro da potência sem a carga aplicada.

Na *Tabela 11,* tem-se o teste aplicado a uma rotação de 1200 RPM.

Tabela 11 - Comparativo de malha fechada com 1200 RPM

Descrição	Medidas sem carga	Medidas com carga
Corrente da fonte	0,6 A	1 A
Potência da fonte	14,4 W	24 W
Tensão da carga	8,4 V	7,4 V

Fonte: O autor (2017)

Nota-se na *Tabela 11*, assim como o teste descrito na *Tabela 10*, a demanda de potência aumenta quando a carga é aplicada, porém a rotação sofre com ligeira oscilação devido a ação do controle P.I, principalmente devido aos valores empíricos utilizados como paramentos de correção KP e KI.

Nota-se também, que em ambos os teste os transistores de chaveamento têm um considerável aumento da temperatura durante a ação do controle P.I., porém sem danifica-los.

4 **CONSIDERAÇÕES FINAIS**

O presente Trabalho descreve o desenvolvimento de um protótipo de controlador para motor sem escovas com imãs permanentes baseado em um microcontrolador dsPIC30F2010.

O primeiro passo para a desenvolvimento do protótipo tem-se na avaliação dos pré-requisitos de funcionamento dos motores de imã permanentes, e a seleção do microcontrolador disponíveis no mercado apto para realizar essa função, bem como os componentes de potência capazes de satisfazer as necessidades de operação do motor adquirido.

Define-se pela utilização de uma placa de controle já desenvolvida, que traz agilidade e modularidade ao desenvolvimento do protótipo, a partir dessa, desenvolve-se o esquemático da placa controladora de potência baseado nas especificações do fabricante do microcontrolador.

Com o esquemático definido, inicia-se a montagem na matriz de contatos do circuito, para a realização dos testes iniciais de interligação e funcionamento dos componentes e o desenvolvimento dos softwares de testes.

Nos testes iniciais com a utilização do motor no circuito desenvolvido na matriz de contatos, nota-se a problemática da elevada quantidade de ruído elétrico gerada sobre a fonte de alimentação quando o motor gira, que tem-se como a principal problemática enfrentada ao longo do desenvolvimento do protótipo, influenciando de forma significativa no desempenho dos componentes projetados, bem como o desenvolvimento de uma placa extra na tentativa de minimizar seus impactos, que só é amenizado com a implementação de filtro nos circuitos afetados, com o aterramento da carcaça do motor junto a fonte, e a alteração dos condutores de alimentação para uma capacidade de corrente mais elevada.

Após o contorno dos efeitos causados pela interferência do giro do motor e atestando o correto funcionamento do circuito projetado, desenvolve-se o layout da placa de protótipo, onde verifica-se as dimensões necessárias para sua construção, destacando-se a ideia de desenvolvimento, onde as questões construtivas da placa tem foco na praticidade na realização das medidas e identificação dos circuitos; possibilidade de alteração ou implementação de recursos de hardware de maneira ágil; opta-se pela utilização de placa perfurada em sua confecção.

Nos testes de avaliação do funcionamento do protótipo construído na placa perfurada, três diferentes rotinas de software tem aplicação ao longo do desenvolvimento com implementação gradual dos recursos, tem-se essa metodologia para assegurar as interdependências dos diferentes circuitos atuantes, sendo a implementação de recursos de maneira gradual minimiza a probabilidade de

alguma lógica errônea impossibilitar a operação de maneira adequada.

Destacam-se nos resultados obtidos dos testes realizados a constante presença de interferências com significância sobre a fonte de tensão, o que interfere consideravelmente nas leituras dos sensores e gera um aquecimento considerável dos transistores de potência quando o motor se encontra em altas rotações, ou em ações constantes do controle P.I.

Com estas considerações é possível ver que os objetivos foram alcançados, sendo possível atender os princípios de funcionamento de maneira satisfatória de um motor sem escovas com imãs permanentes com a utilização de um microcontrolador dsPIC30F2010

Como sugestão para trabalhos futuros destaca-se a possibilidade de aprimoramento do projeto, principalmente com a análise dos efeitos das interferências com diferentes motores e transistores de potência responsáveis pelo chaveamento, e também, a implementação de um controle P.I.D para alguma aplicação. O desenvolvimento de uma placa profissional com tecnologia *SMD* pode tornar o projeto viável para implementação comercial, ou ainda o desenvolvimento com carácter acadêmico, para estudos aprofundados em diferentes implementações.

REFERÊNCIAS

ALLEGRO MICROSYSTEMS. **ACS712**. 2017. 16 p. Disponível em <http://www.allegromicro.com/~/media/Files/Datasheets/ACS712-Datasheet.ashx>. Acesso em: 31 Out 2017.

ALLTRONICS. **Lin Engineering**: High Speed/Torque BRUSHLESS DC MOTOR - 28M126. 2013. 3 p. Disponível em <http://www.alltronics.com/mas_assets/acrobat/28M126.pdf>. Acesso em: 15 Abr 2017.

GUSIDOETA, Ivan V.; CAPUANO, Francisco G. **Elementos de eletrônica digital**. 40. ed. São Paulo, SP: Érica, 2008.

GUSSOW, Milton. **Eletricidade básica**. 2. ed. Porto Alegre, RS: Bookman, 2009. 571 p. (Schaum)

INFINEON TECHNOLOGIES. **HV Floating MOS-Gate Driver ICs**. 2007. 30 p. Disponível em <http://www.infineon.com/dgdl/an-978.pdf>. Acesso em: 15 Abr 2017.

_____. **IR2101(S)/IR2102(S) & (PbF)**: HIGH AND LOW SIDE DRIVER. 2004. 14 p. Disponível em <https://www.infineon.com/dgdl/ir2101.pdf?fileId=5546d462533600a4015355c7a755166c>. Acesso em: 31 Out 2017.

_____. **IRF530N**: HEXFET Power MOSFET. 2001. 16 p. Disponível em <https://www.infineon.com/dgdl/irf530n.pdf?fileId=5546d462533600a4015355e380521998>. Acesso em: 31 Out 2017.

INSTITUTO NEWTON C BRAGA. **Indutores, capacitores e filtros (EL001)**. Disponível em <http://www.newtoncbraga.com.br/index.php/144-eletrotecnica-e-eletricidade/eletricidade-industrial-eletrotecnica/1814-indutores-capacitores-e-filtros-el001>. Acesso em: 13 Nov 2017.

INSTITUTO NEWTON C. BRAGA. **Como funcionam os sensores de Efeito Hall**: (ART1050). Disponível em <http://www.newtoncbraga.com.br/index.php/como-funciona/6640-como-funcionam-os-sensores-de-efeito-hall-art1050>. Acesso em: 22 Abr 2017.

_____. **Reguladores de tensão 7800 (ART156)**. Disponível em <http://www.newtoncbraga.com.br/index.php/como-funciona/1076-art156>. Acesso em: 13 Nov 2017.

LPKF LASER & ELECTRONICS. **LPKF TechGuide**: In-House PCB Prototyping. 2016. 40 p. Disponível em <http://www.lpkf.com/_mediafiles/4080-techguide-in-house-pcb-prototyping-en.pdf>. Acesso em: 09 Nov 2017.

MICROCHIP. **Brushless DC (BLDC)**: Motor Fundamentals. 2003. 20 p. Disponível em <http://ww1.microchip.com/downloads/en/AppNotes/00885a.pdf>. Acesso em: 15 Abr 2017.

_____. **Brushless DC Motor Control Made Easy**. 2011. 48 p. Disponível em <http://ww1.microchip.com/downloads/en/AppNotes/00857B.pdf>. Acesso em: 15 Abr 2017.

_____. **dsPIC30F Family Reference Manual**. 2005. 760 p. Disponível em <http://ww1.microchip.com/downloads/en/DeviceDoc/70046D.pdf>. Acesso em: 13 Nov 2017.

_____. **dsPIC30F/33F Programmer's Reference Manual**. 2008. 364 p. Disponível em <http://ww1.microchip.com/downloads/en/DeviceDoc/70157C.pdf>. Acesso em: 13 Nov 2017.

_____. **dsPIC30F2010 Data Sheet**: 28-pin High-Performance Digital Signal Controllers. 2004. 202 p. Disponível em <http://ww1.microchip.com/downloads/en/devicedoc/70118e.pdf>. Acesso em: 15 Abr 2017.

_____. **PICkit™ 3**: Programmer/Debugger User's Guide. 2010. 100 p. Disponível em <http://ww1.microchip.com/downloads/en/DeviceDoc/51795B.pdf>. Acesso em: 27 Mai 2017.

_____. **Sensored BLDC Motor Control Using dsPIC30F2010**. 2004. 20 p. Disponível em <http://ww1.microchip.com/downloads/en/AppNotes/BLDC%20MC%2000957a.pdf>. Acesso em: 15 Abr 2017.

_____. **Sensorless BLDC Motor Control Using dsPIC30F2010**. 2005. 16 p. Disponível em <http://ww1.microchip.com/downloads/cn/AppNotes/cn554233.pdf>. Acesso em: 15 Abr 2017.

_____. **Using the dsPIC30F for Sensorless BLDC Control**. 2004. 36 p. Disponível em <http://ww1.microchip.com/downloads/en/AppNotes/Sensorless%20BLDC%20AN901a.pdf>. Acesso em: 15 Abr 2017.

_____. **VF Control of 3-Phase Induction Motors**: Using PIC16F7X7 Microcontrollers. 2004. 24 p. Disponível em <http://ww1.microchip.com/downloads/en/AppNotes/00889b.pdf>. Acesso em: 15 Abr 2017.

NATIONAL SEMICONDUCTOR. **LM78XX Series Voltage Regulators**. 1995. 6 p. Disponível em <http://www.hep.upenn.edu/SNO/daq/parts/lm7815.pdf>. Acesso em: 31 Out 2017.

OLIVEIRA, André Schneider de; ANDRADE, Fernando Souza de. **Sistemas embarcados**: hardware e firmware na prática. 1. ed. São Paulo, SP: Érica, 2006.

316 p.

PEREIRA, Fábio. **Microcontroladores PIC**: programação em C. 7. ed. São Paulo: Érica, 2007. 358 p.

WEG. **Motores de corrente contínua**: Manual de Instalação, Operação e Manutenção. 2012. 63 p. Disponível em <http://ecatalog.weg.net/files/wegnet/WEG-motor-de-corrente-continua-10218369-manual-portugues-br.pdf>. Acesso em: 13 Nov 2017.

_____. **Motores Elétricos**: Guia de especificação. 20. ed. Jaraguá do Sul, 2016. 68 p. Disponível em <http://ecatalog.weg.net/files/wegnet/WEG-guia-de-especificacao-de-motores-eletricos-50032749-manual-portugues-br.pdf>. Acesso em: 05 Jun 2017.

XIA, Chang-liang. **Permanent Magnet Brushless DC Motor Drives and Controls**. 1. ed. John Wiley & Sons Singapore Pte. Ltd, 2012. 320 p.

APÊNDICE A- Código-fonte temporal

Este apêndice contém o código-fonte desenvolvido para testes iniciais, com rotação através de interrupção de *timer 1* e controle de largura de pulso por potenciômetro.

```
/*
Autor: Leandro Borges - Universidade Feevale
Data: 09/11/2017
Titulo: Teste de Ligações Placa Protótipo na matriz de contatos
Ver: V1.0 APÊNDICE A
Assunto:01 - LED1 de atividade na saida RC13
02 - Botão de Start/Stop na entrada RC14
03 - Teste MCPWM
04 - Timer 1 controla velocidade de rotação
05 - Potenciômetro controla PWM
05 - Leitura no osciloscópio do sensor Hall CN5,6 e 7 pullups externo
---------------------------------------------------------------------------
Mapeamento das ligações do DSPIC30F2010 aos outros módulos
---------------------------------------------------------------------------
Pino Função Pino Ligação dos pinos aos módulos utilizados
---------------------------------------------------------------------------
01 MCLR Reset e pino 6 para gravação in-circuit
02 EMUD3/AN0/VREF+/CN2/RB0 J4-1 Ch1 = AN0 = V24
03 EMUC3/AN1/VREF-/CN3/RB1 J4-2 Ch2 = AN1 = IMotor
04 AN2/SS1/CN4/RB2 J4-3 Ch0 = AN2 = Potenciometro
05 AN3/INDX/CN5/RB3 J4-4 Hall A
06 AN4/QEA/IC7/CN6/RB4 J4-5 Hall B
07 AN5/QEB/IC8/CN7/RB5 J4-6 Hall C
08 VSS j4-7 e J6-1 Terra
09 OSC1/CLK1 Cristal 20MHz para clock
10 OSC2/CLKO/RC15 Cristal 20MHz para clock
11 EMUD1/SOSC1/T2CK/U1ATX/CN1/RC13 J6-8 - Led de atividade, 1K + Led + gnd
12 EMUC1/SOSC0/T1CK/U1ARX/CN0/RC14 J6-7 - Botão Start/Stop Press Close
13 VDD j4-8 VDD 5,00V
14 PREMUD2/OC2/IC2/INT2/RD1 J6-4 NC
15 EMUCS/OC1/IC1/INT1/RD0 J6-5 NC
16 FLTA/INT0/SCK1/OCFA/RE8 J6-6 - Jumper de Direção(NO) => "0" Rotacao Direita
17 PGD/EMUD/U1TX/SDO1/SDA/RF3 UART1 ligada ao Max 232 com jumper JP4
18 PGC/EMUC/U1RX/SDI1/SDA/RF2 UART1 ligada ao Max 232 com jumper JP3
19 VSS J6-1 e J4-7 Ground
20 VDD J4-8 VDD 5,00V
21 PWM3H/RE5 J5-1 PWM5 - Fase C Preto
22 PWM3L/RE4 J5-2 PWM4 - Fase C Preto
23 PWM2H/RE3 J5-3 PWM3 - Fase B Vermelho
24 PWM2L/RE2 J5-4 PWM2 - Fase B Vermelho
```

```
25 PWM1H/RE1 J5-5 PWM1 - Fase A Amarelo
26 PWM1L/RE0 J5-6 PWM0 - Fase A Amarelo
27 AVSS Terra virtual
28 AVDD VCC virtual
-------------------------------------------------------------------------*/
//*****************************************************************************************
//***** Bibliotecas Utilizadas ************************************************************
//*****************************************************************************************
#include<p30f2010.h> //Biblioteca do dsPIC30F2010
//*****************************************************************************************
//***** Fuses *****************************************************************************
//*****************************************************************************************
_FOSC(CSW_FSCM_OFF & HS); // Oscilador externo sem tratamento de erro
_FWDT(WDT_OFF); // WatchDog disabilitado
_FBORPOR(MCLR_EN & PBOR_OFF & PWRT_OFF); // Habilita MCLR, Desabilita PBOR e PWRT
_FGS(CODE_PROT_OFF); // Sem protecao de codigo - é possível a leitura
// posterior a gravação do chip
//*****************************************************************************************
//*****************************************************************************************
/*
Type Bits Min Max
char, signed char 8 -128 127
unsigned char 8 0 255
short, signed short 16 -32768 32767
unsigned short 16 0 65535
int, signed int 16 -32768 32767
unsigned int 16 0 65535
long, signed long 32 -2^31 2^(31 - 1)
unsigned long 32 0 232 - 1
long long**, signed long long** 64 -2^63 2^(63 - )1
unsigned long long** 64 0 2^(64 - 1)
Type Bits E Min E Max N Min N Max
float 32 -126 127 2^(-126) 2^128
double* 32 -126 127 2^(-126) 2^128
long double 64 -1022 1023 2^(-1022) 2^1024
*/
//*****************************************************************************************
//**** Define Valores Padrões para o Programa *********************************************
//*****************************************************************************************
#define Fcy 5000000 //Freqüência do ciclo=freqüência do oscilador/4
//dsPIC necessita de 4 ciclos de clock para
// executar um ciclo de instrução
#define FPWM 16000 //Frequência MCPWM
#define LED1 LATCbits.LATC13 //Define LED1 no pino RC13
#define S2 PORTCbits.RC14 //Define Botão Start/Stop no pino RC14
//*****************************************************************************************
//******** Declarando as variáveis ********************************************************
//*****************************************************************************************
```

```
unsigned int StateLoTable[] =
{0x0000, 0x0210, 0x2004, 0x0204, 0x0801, 0x0810, 0x2001, 0x0000}; //Rotacao esquerda
unsigned char nHVLoTable[] =
{0x00, 0x05, 0x04, 0x06, 0x02, 0x03, 0x01, 0x000}; //Leitura Sensor Hall Manual
unsigned int StateRoTable[] =
{0x0000, 0x2001, 0x0810, 0x0801, 0x0204, 0x2004, 0x0210, 0x0000}; //Rotacao direita
unsigned char nHVRoTable[] =
{0x00, 0x01, 0x03, 0x02, 0x06, 0x04, 0x05, 0x000}; //Leitura Sensor Hall Manual
unsigned int HallValue = 1;
//*****************************************************************************************
//************ Funções Utilizadas*********************************************************
//*****************************************************************************************
void InitMCPWM(void); //Declara a função iniciar PWM
void InitADC10(void); //Declara a função iniciar ADC
void inicia_timer1(void); //Declara a função iniciar Timer1
void __attribute__((interrupt, auto_psv))_ADCInterrupt(void); //Declara a função interrupção ADC
void __attribute__((interrupt, auto_psv)) _T1Interrupt( void ); //Declara a função interrupção T1
//*****************************************************************************************
//*************************** Rotina Principal do Programa ********************************
//*****************************************************************************************
int main(void)
{
TRISD = 0xffff; //RD0 e RD1 p/ entrada. Não utilizada
TRISC = 0xdfff; //RC13 (LED1) p/ saída e RC14 (Botão) p/ entrada
TRISE = 0xFFC0; //RE0, 1, 2, 3, 4 e 5 p/ saída dos PWMs
CNEN1 = 0x00E0; //CN5,6 e 7 ativadas (Sensor Hall)
CNPU1 = 0x0000; //Desativa pullups internos do Processador
inicia_timer1(); //Inicializa o Timer 1
InitADC10(); //Inicializa a conversão A/D
InitMCPWM(); //Inicializa o MCPWM
while(1) //Loop infinito
{
while(S2) //Botão não pressionado
{
IEC0bits.T1IE = 0; //Desabilita a interrupção do timer1
PWMCON1 = 0x0700; //Desabilita saídas do PWM
//Motor não Gira
}
while(S2==0) //Quando o Botão Pressionado
IEC0bits.T1IE = 1; //Habilita a interrupção do timer1
} //Motor Gira
}
//*****************************************************************************************
//********************** Tratamento interrupção Timer 1 ***********************************
//*****************************************************************************************
void __attribute__((interrupt, auto_psv)) _T1Interrupt( void )
{
LED1 = !LED1; //Pisca o LED 1 Conforme o Giro
```

```
if (HallValue < 6) //Alteração dos valores de Hall
HallValue = HallValue+1; // manualmente, sem leitura dos
else HallValue=1; // sensores.
OVDCON = StateRoTable[nHVRoTable[HallValue]];
//Carrega registrador de sobrepor
PDC1 = ADCBUF0/4; //Utiliza Valor do potênciomentro
PDC2 = PDC1; // para controlar o PWM igualmente
PDC3 = PDC1; // em todas as Fases.
PWMCON1 = 0x0777; //Habilita saídas do PWM
IFS0bits.T1IF = 0; //Reseta a flag de interrupção
}
//*************************************************************************************************
//************ Tratamento interrupção Conversor Analogico Digital *********************************
//*************************************************************************************************
void __attribute__((interrupt, auto_psv))_ADCInterrupt(void)
{
IFS0bits.ADIF = 0; //Leitura do Potenciomentro
}
//*************************************************************************************************
//****** Rotina de inicialização do timer 1 - Velocidade motor************************************
//*************************************************************************************************
void inicia_timer1(void)
{
T1CON = 1; //Desabilita a contagem do timer1
TMR1 = 0; //Zera a contagem de tempo do timer1
IFS0bits.T1IF = 0; //Clear no bit de chamada da interrupção do timer1
PR1 = Fcy/256;//128;//512; //Ajusta o período de tempo do timer1 para o valor da freqüência
// do ciclo de instrução/256 = 1 segundo
// ou seja, a rotação do motor
T1CONbits.TCKPS = 1; //Ajusta o prescaler
IPC0bits.T1IP = 7; //Define a prioridade da interrupção do timer1=7
IEC0bits.T1IE = 1; //Habilita a interrupção do timer1
T1CONbits.TON = 1; //1 = habilita a contagem do timer1
}
//*************************************************************************************************
//***** Inicialização do módulo MCPWM *************************************************************
//*************************************************************************************************
void InitMCPWM(void)
{
PTPER = Fcy/FPWM-1; //Base de tempo para o periodo do registrador
PWMCON1 = 0x0700; //Desativa os PWMs
OVDCON = 0x0000; //Permite o controle Usando o Registrador OVDCON
PWMCON2 = 0x0F00; //Habilita postscale de 16 vezes
PTCON = 0x8000; //Inicializa o PWM
}
//*************************************************************************************************
//***** Inicialização do módulo ADC de 10 Bits *****************************************************
//*************************************************************************************************
```

```
void InitADC10(void)
{
ADPCFG = 0xFFF8; //PORTB = Digital exceção de RB0, RB1 e RB2 = analógico
ADCON1 = 0x0064; //Ativa o MCPWM nas saidas PWM
ADCON2 = 0x0200; //Amostra Canal 0 em AN2
ADCON3 = 0x0080; //Tad = internal RC (4uS)
ADCHS = 0x0002; //Canal 0 controla PWM.
IFS0bits.ADIF = 0; //Limpa a flag de interrupção do AD
IEC0bits.ADIE = 1; //Habilita a interrupção do AD
ADCON1bits.ADON = 1;//Liga o AD.
}
//*****************************************************************************************
//********************************* Fim ***************************************************
//*****************************************************************************************
```

APÊNDICE B- Código-fonte de malha aberta

Este apêndice contém o código-fonte desenvolvido para testes iniciais de malha aberta.

```
/*
Autor: Leandro Borges - Universidade Feevale
Data: 15/11/2017
Titulo: Teste de Placa Protótipo
Ver: V1.1 APÊNDICE B
Assunto:01 - LED1 de atividade na saida RC13
02 - Botão de Start/Stop na entrada RC14
03 - MCPWM
04 - OpenLoop nas saídas do PWM
05 - Start/Stop
06 - Controle de velocidade pelo potenciômentro
07 - Sensor Hall habilitado em CN5,6 and 7
08 - Verificação tensão de 24V
09 - Verificação de corrente
10 - Cálculo de Velocidade
11 - Transmissão via UART
--------------------------------------------------------------------------
Mapeamento das ligações do DSPIC30F2010 aos outros módulos
--------------------------------------------------------------------------
Pino Função Pino Ligação dos pinos aos módulos utilizados
--------------------------------------------------------------------------
01 MCLR Reset e pino 6 para gravação in-circuit
02 EMUD3/AN0/VREF+/CN2/RB0 J4-1 Ch1 = AN0 = V24
03 EMUC3/AN1/VREF-/CN3/RB1 J4-2 Ch2 = AN1 = IMotor
04 AN2/SS1/CN4/RB2 J4-3 Ch0 = AN2 = Potenciômetro
05 AN3/INDX/CN5/RB3 J4-4 Hall A
06 AN4/QEA/IC7/CN6/RB4 J4-5 Hall B
07 AN5/QEB/IC8/CN7/RB5 J4-6 Hall C
08 VSS j4-7 e J6-1 Terra
09 OSC1/CLK1 Cristal 20MHz para clock
10 OSC2/CLKO/RC15 Cristal 20MHz para clock
11 EMUD1/SOSC1/T2CK/U1ATX/CN1/RC13 J6-8 - Led de atividade, 1K + Led + gnd
12 EMUC1/SOSC0/T1CK/U1ARX/CN0/RC14 J6-7 - Botão Start/Stop
13 VDD j4-8 VDD 5,00V
14 PREMUD2/OC2/IC2/INT2/RD1 J6-4 NC
15 EMUCS/OC1/IC1/INT1/RD0 J6-5 NC
16 FLTA/INT0/SCK1/OCFA/RE8 J6-6 - Jumper de Direção(NO) => "0" Rotacao Direita
17 PGD/EMUD/U1TX/SDO1/SDA/RF3 UART1 ligada ao Max 232 com jumper JP4
18 PGC/EMUC/U1RX/SDI1/SDA/RF2 UART1 ligada ao Max 232 com jumper JP3
19 VSS J6-1 e J4-7 Ground
20 VDD J4-8 VDD 5,00V
```

```
21 PWM3H/RE5 J5-1 PWM5 - Fase C Preto
22 PWM3L/RE4 J5-2 PWM4 - Fase C Preto
23 PWM2H/RE3 J5-3 PWM3 - Fase B Vermelho
24 PWM2L/RE2 J5-4 PWM2 - Fase B Vermelho
25 PWM1H/RE1 J5-5 PWM1 - Fase A Amarelo
26 PWM1L/RE0 J5-6 PWM0 - Fase A Amarelo
27 AVSS Terra virtual
28 AVDD VCC virtual
-------------------------------------------------------------------------
-------------------------------------------------------------------------
*/
//*****************************************************************************************************
//***** Bibliotecas Utilizadas ************************************************************************
//*****************************************************************************************************
#include<p30f2010.h> //Biblioteca do dsPIC30F2010
#include<uart.h> //Biblioteca UART.h
#include<pwm.h> //Biblioteca PWM.h
#include<reset.h> //Biblioteca reset.h
//*****************************************************************************************************
//***** Fuses *****************************************************************************************
//*****************************************************************************************************
_FOSC(CSW_FSCM_OFF & HS); // Oscilador externo sem tratamento de erro
_FWDT(WDT_OFF); // WatchDog disabilitado
_FBORPOR(MCLR_EN & PBOR_OFF & PWRT_OFF); // Habilita MCLR, Desabilita PBOR e PWRT
_FGS(CODE_PROT_OFF); // Sem protecao de codigo - é possível a leitura
// posterior a gravação do chip
//*****************************************************************************************************
//*****************************************************************************************************
/*
Type Bits Min Max
char, signed char 8 -128 127
unsigned char 8 0 255
short, signed short 16 -32768 32767
unsigned short 16 0 65535
int, signed int 16 -32768 32767
unsigned int 16 0 65535
long, signed long 32 -2^31 2^(31 - 1)
unsigned long 32 0 232 - 1
long long**, signed long long** 64 -2^63 2^(63 - )1
unsigned long long** 64 0 2^(64 - 1)
Type Bits E Min E Max N Min N Max
float 32 -126 127 2^(-126) 2^128
double* 32 -126 127 2^(-126) 2^128
long double 64 -1022 1023 2^(-1022) 2^1024
*/
//*****************************************************************************************************
//**** Define Valores Padrões para o Programa *********************************************************
//*****************************************************************************************************
```

```
#define Fcy 5000000 //Freqüência do ciclo=freqüência do oscilador/4
//dsPIC necessita de 4 ciclos de clock para
// executar um ciclo de instrução
#define FPWM 16000 //Frequência MCPWM
#define NS 0x55 //Define a constante para ID de comunicacao
#define RPMConstant 60*(Fcy/256) //Define Constante 60*10000000/256 = 234375
// cálculo da velocidade
#define LED1 LATCbits.LATC13 //Define LED1 no pino RC13
#define S2 PORTCbits.RC14 //Define Botão Start/Stop no pino RC14
#define S3 PORTEbits.RE8 //Define Juper de direção da rotacao
#define V24 ADCBUF1 //Define V24 na porta RB0:
// verifica valor de tensão de 24V no circuito
#define IMotor ADCBUF2 //Define IMotor na porta RB0
// verifica valor da corrente atravéz do sensor de corrente
//*****************************************************************************************
//******** Declarando as variáveis ***********************************************************
//*****************************************************************************************
unsigned int StateLoTable[] = //Sentido de rotação esquerda
{0x0000, 0x0210, 0x2004, 0x0204, 0x0801, 0x0810, 0x2001, 0x0000};
unsigned int StateRoTable[] = //Sentido de rotação direita
{0x0000, 0x2001, 0x0810, 0x0801, 0x0204, 0x2004, 0x0210, 0x0000};
unsigned int i; //Variável de controle de loops
unsigned int HallValue = 1; //Variável sensor hall
unsigned char RunMotor = 0; //Flag para start/stop 0 = stop
unsigned char Direction = 0; //Flag para sentido de rotacao 0 = esquerda
int Speed; //Variável de velocidade
unsigned int timer3value; //Variável Valor de Timer 3
unsigned int timer3avg; //Variável com a média do valor de Timer 3
unsigned char polecount; //Variável da contagem dos polos do motor
unsigned char TxData = 0; //Variável para envio do pacote de dados via uart
unsigned char S1 = 0; // Start/Stop via uart
unsigned char S4 = 1; // Sentido de rotacao via uart
int DesiredPosition = 0;
//*********Protocolo do pacote de comunicação = Head, byte1, byte2, Comando ***********************
unsigned char Header; //define a variável caracter Header
unsigned char B1Tx; //define a variável para o Byte 1 do pacote de transmissao
unsigned char B2Tx; //define a variável para o Byte 2 do pacote de transmissao
unsigned char B1Rx; //define a variável para o Byte 1 do pacote de recepcao
unsigned char B2Rx; //define a variável para o Byte 2 do pacote de recepcao
unsigned char CmdRx; //define a variável para o Comando a ser executado
unsigned char CmdTx; //define a variável para Comando a ser transmitido
//*****************************************************************************************
//************* Funções Utilizadas*********************************************************
//*****************************************************************************************
//***********************************************************************
void delayms(int d){ double i; int f; f = 13*d;
for (i=0;i<f;i++){} } // Para d = 1, atraso de tempo de 1 mS
void InitMCPWM(void); //Declara a função iniciar PWM
```

```
void InitADC10(void); //Declara a função iniciar ADC
void inicia_timer1(void); //Declara a função iniciar Timer1
void InitTMR3(void); //Declara a função iniciar Timer3
void __attribute__((interrupt, auto_psv))_ADCInterrupt(void); //Declara a função interrupção ADC
void __attribute__((interrupt, auto_psv)) _T1Interrupt( void ); //Declara a função interrupção T1
void inicia_UART(void); //Declara a função iniciar UART1
void __attribute__((interrupt, auto_psv))_U1TXInterrupt(void); //Interrupção de transmissão de dados
void __attribute__((interrupt, auto_psv))_U1RXInterrupt(void); //Interrupção de recepção de dados
void TransPac (void); //Declara a função para transmitir um pacote de 4 bytes via UART
void Start(void); //Declara a função Start, que retira o motor do repouso
void Stop(void); //Declara a função Stop, que coloca o motor em repouso
//*****************************************************************************************************
//***** Rotina Principal do Programa *****////////////////////////////////////////////////////
//*****************************************************************************************************
int main(void)
{
TRISD = 0xffff; //RD0 e RD1 p/ entrada. Não utilizada
TRISC = 0xdfff; //RC13 (LED1) p/ saída e RC14 (Botão) p/ entrada
TRISE = 0xFFC0; //RE0, 1, 2, 3, 4 e 5 p/ saída dos PWMs
CNEN1 = 0x00E0; //CN5,6 e 7 ativadas (Sensor Hall)
CNPU1 = 0x0000; //Desativa pullups internos do Processador
IFS0bits.CNIF = 0; //Limpa o registrador de interrupção (flag)
IEC0bits.CNIE = 1; //Ativa a interrupção CN
inicia_UART(); //Inicializa UART
inicia_timer1(); //Inicializa o Timer 1
InitTMR3(); //Inicializa o Timer 3
InitADC10(); //Inicializa a conversão A/D
InitMCPWM(); //Inicializa o MCPWM
while(1){ //Loop Infinito
while(S2) //Botão não pressionado
{
if( RunMotor == 1 && IMotor > 0x3FF)
{ //Se corrente maior que 5A
Stop(); //Para o motor e
for (i=0; i<10; i++) //Pisca o LED 10 vezes.
{LED1=!LED1; delayms(100);}
}
}
while(!S2) //Mudança de estado botão
{
if( RunMotor == 1) //Se o motor rodando
Stop(); //Desativa o motor
else //Se motor parado
{ //Se tensão entre 12V e 24V
if (V24 > 0x1FF && V24 < 0x3F0)
Start(); //Ativa o motor
else //Se não
{
```

```
for (i=0; i<5; i++) //Pisca o LED 5 vezes.
{
LED1=!LED1;
delayms(100);
}}}}}}
void Start(void)
{
while(S2) //Aguarda liberação do botão
{
HallValue = PORTB & 0x0038; //Mascara de bits em RB3,4 & 5
HallValue = HallValue >> 3; //Deslocamento de bits 3 vezes
Direction = S3; //Direção de rotação
if (Direction == 1) //Se Direção 1 Rotação para direita
OVDCON = StateRoTable[HallValue];//Carrega o registrador de sobreposição
else //Se direção 0, Rotação para esquerda
OVDCON = StateLoTable[HallValue];//Carrega o registrador de sobreposição
PDC1 = 0x500; //Inicia com PWM de valor elevado
PDC2 = PDC1; // devido a inércia.
PDC3 = PDC1;
PWMCON1 = 0x0777; //Habilita as saídas PWM
delayms(10); //Aguarda 10ms
LED1 = 1; //Ativa o LED1
RunMotor = 1; //Altera flag para motor rodando
goto fim; //Sai do laço While.
}
fim:;
}
void Stop(void)
{
while(S2) //Aguarda liberação do botão
{
PWMCON1 = 0x0700; //Desabilita as saidas PWM
OVDCON = 0x0000; //Desabilita o Registrador de sobreposição
delayms(10); //Aguarda 10ms
RunMotor = 0; //Altera flag para motor rodando
LED1 = 0; //Ativa o LED1
goto fim; //Sai do laço While.
}
fim:;
}
//*****************************************************************************************************
//********************** Tratamento interrupção Timer 1 ***********************************************
//*****************************************************************************************************
void __attribute__((interrupt, auto_psv)) _T1Interrupt( void )
{
IFS0bits.T1IF = 0; // reset timer interrupt flag
TransPac();
if( RunMotor == 0 && i==30 ) //motor parado
```

```
{
LED1=!LED1; // LED pisca 5 vezes
i=0;
}
else i++;
}
void __attribute__((interrupt, auto_psv))_CNInterrupt(void)
{
IFS0bits.CNIF = 0; //Reseta a flag de interrupção
HallValue = PORTB & 0x0038; //Mascara em RB3,4 & 5
HallValue = HallValue >> 3; //Deslocamento para direita 3 vezes
if (Direction == 1)
OVDCON = StateRoTable[HallValue]; //Carrega registrador de sobrepor
else OVDCON = StateLoTable[HallValue]; //Carrega registrador de sobrepor
//Cálculo da velocidade.
if (HallValue == 1) //A mesma posição já foi verificada?
if (polecount++ == 2) //Dois pares de polos, uma volta
{
timer3value = TMR3; //Guarda o valor de Timer 3
TMR3 = 0; //Zera o Timer 3
timer3avg = ((timer3avg + timer3value) >> 1);//Calcula a média,
polecount = 1; //Reseta a contagem dos polos
Speed = RPMConstant/timer3avg; //Calcula a velocidade
}
}
//*********************************************************************************************
//************ Tratamento interrupção Conversor Analogico Digital *********************************
//*********************************************************************************************
void __attribute__((interrupt, auto_psv))_ADCInterrupt(void)
{
IFS0bits.ADIF = 0; //Leitura do Potenciomentro
PDC1 = ADCBUF0/3; //Atribui os valor do potenciomentro
PDC2 = PDC1; // a todas as três saídas do PWM
PDC3 = PDC1; // controla o tempo de trabalho
}
//*********************************************************************************************
//************************ Rotina de inicialização do timer 1 ************************************
//*********************************************************************************************
void inicia_timer1(void)
{
T1CON = 1; //Desabilita a contagem do timer1
TMR1 = 0; //Zera a contagem de tempo do timer1
IFS0bits.T1IF = 0; //Clear no bit de chamada da interrupção do timer1
PR1 = Fcy/512;//128;//256; //Ajusta o período de tempo do timer1 para o valor da freqüência
// do ciclo de instrução/256 = 1 segundo
T1CONbits.TCKPS = 1; //Ajusta o prescaler
IPC0bits.T1IP = 7; //Define a prioridade da interrupção do timer1 = 7
IEC0bits.T1IE = 1; //Habilita a interrupção do timer1
```

```
T1CONbits.TON = 1; //1 = habilita a contagem do timer1
}
//*********************************************************************************************
//************************ Rotina de inicialização do timer 3 ***********************************
//*********************************************************************************************
void InitTMR3(void)
{
T3CON = 0x0030; // internal Tcy/256 clock
TMR3 = 0;
PR3 = 0xffff; //Fcy/256; //
T3CONbits.TON = 1; //1 = habilita a contagem do timer3
}
//*********************************************************************************************
//***** Inicialização do módulo ADC de 10 Bits ************************************************
//*********************************************************************************************
void InitADC10(void)
{
ADPCFG = 0xFFF8; // PORTB = Digital exceção de RB0, RB1 e RB2 = analógico
ADCON1 = 0x0064; // Ativa o MCPWM nas saidas PWM 1100100
ADCON2 = 0x030C; // Amostragem simultanea dos 3 canais analogicos
ADCON3 = 0x0302; // Utilização do clock interno Tad = internal 2*Tcy
ADCHS = 0x0002; // Conecta RB2/AN2 no CH0
IFS0bits.ADIF = 0; // Limpa a flag de interrupção do AD
IEC0bits.ADIE = 1; // Habilita a interrupção do AD
ADCON1bits.ADON = 1;// Liga o AD.
}
//*********************************************************************************************
//***** Inicialização do módulo MCPWM *********************************************************
//*********************************************************************************************
void InitMCPWM(void)
{
PTPER = Fcy/FPWM - 1; //Base de tempo para o periodo do registrador
PWMCON1 = 0x0700; //Desativa os PWMs
OVDCON = 0x0000; //Permite o controle usando o registrador OVDCON
PDC1 = 500; //PWM Inicial do Motor, Antes da Leitura do POT
PDC2 = 500;
PDC3 = 500;
SEVTCMP = PTPER; //Habilita a interrupção especial do ADC
PWMCON2 = 0x0F00; //Habilita postscale de 16 vezes
PTCON = 0x8000; //Inicializa o PWM
}
//*********************************************************************************************
//***** Inicialização do módulo UART **********************************************************
//*********************************************************************************************
void inicia_UART()
{
unsigned int ubrg; //Define a variável ubrg
unsigned int config1; //Define a variável config1
```

```
unsigned int config2; //Define a variável config2
CloseUART1(); //Desabilita a UART1
ubrg = 32; //Ubrg=(fcy/(BaudRate*16))-1
//Fcy=FCristal/4
//Neste Caso: Baud Rate=9600
//Freq.Cristal=20MHz
//Fcy=5MHz, ubrg=31,55
ConfigIntUART1(UART_RX_INT_EN & //Função para configuração das interrupções da UART
//Habilita a interrupção de recepção
UART_RX_INT_PR2 & //Define a prioridade da interrupção de recepção=2
UART_TX_INT_DIS & //Habilita a interrupção de transmissão
UART_TX_INT_PR4); //Define a prioridade da interrupção de transmissão=4
config1 = UART_EN & //Define a variável config1 com os seguintes parâmetros
//Habilita o módulo UART
UART_IDLE_CON & //UART funcionando no modo IDLE
UART_RX_TX & //Configura a comunicação através dos pinos padrões
// de TX e RX (U1TX e U1RX)
UART_DIS_WAKE & //Desabilita o modo "Wake-up"
UART_DIS_LOOPBACK & //Desabilita o modo "loop back"
UART_DIS_ABAUD & //Desabilita o modo "autobaud"
UART_NO_PAR_8BIT & //Define a comunicação com 8 bits de dados e
// sem bit de paridade
UART_1STOPBIT; //Define o Stop Bit=1
config2 = UART_INT_TX_BUF_EMPTY &
//Define a variável config2 com os seguintes parâmetros
//Ocorre interrupção de transmissão quando buffer de
// transmissão for esvaziado
UART_TX_PIN_NORMAL &//Define o bit TXBreak como normal
UART_TX_ENABLE & //Habilita a transmissão da UART
UART_INT_RX_BUF_FUL &
//Ativa a interrupção de recepção para Buffer cheio
UART_ADR_DETECT_DIS &
//Desativa o modo de "adress detect"
UART_RX_OVERRUN_CLEAR;
//Limpa o flag de "overrun"
OpenUART1(config1, config2, ubrg); //Habilita a UART1 com os parâmetros de configuração
// definidos pelas variáveis Config1, config2 e ubrg
}
//*****************************************************************************************
//********* Rotina de tratamento da Interrupção da recebimento de dados ***************************
//*****************************************************************************************
void _ISR _U1RXInterrupt(void){ //Função da rotina de interrupção de recepção de dados
TMR1=0; //Zera a contagem do Timer 1
//Protocolo de comunicação = Head, byte1, byte2, Comando
while(BusyUART1()); //Retorna o status da transmissão;
//Espera enquanto a UART estiver ocupada
Header=ReadUART1(); //Lê o primeiro byte do buffer de recepção da UART = NS
B1Rx=ReadUART1(); //Lê o segundo byte do buffer de recepção da UART = Byte 1
```

```
B2Rx=ReadUART1(); //Lê o terceiro byte do buffer de recepção da UART = Byt 2
CmdRx=ReadUART1(); //Lê o quarto byte do buffer de recepção da UART = Comando
if(Header==NS){ //Compara se número de série recebido é igual ao correto(NS)
//Se for igual executa comando
DesiredPosition = B1Rx * 255 + B2Rx;
if (CmdRx==0){ S1 = 0;} // Stop
if (CmdRx==1) { S1 = 1;} // Start
//Se for diferente ignora e retorna para a rotina principal
}
IFS0bits.U1RXIF=0; //Clear no bit de chamada da interrupção de recepção
} //Fim da interrupção de recepção da UART
//*********************************************************************************************
//***************Rotina para enviar o pacote de dados via UART ***********************************
//*********************************************************************************************
void TransPac (void)
{ //Função que transmite o pacote de dados
switch(TxData & 0x03){//TxData & 0x03
case 0x00: //caso Cmd=0x00 então executa os seguintes comandos
Header = NS;
B1Tx = ADCBUF0 >> 8; //Envia Valor do Potenciômetro
B2Tx = ADCBUF0 & 0x00FF;
CmdTx = 0x00;
break;
case 0x01:
Header = NS;
B1Tx = ADCBUF1 >> 8; //Envia Valor da fonte de Tensão
B2Tx = ADCBUF1 & 0x00FF;
CmdTx = 0x01;
break;
case 0x02:
Header = NS;
B1Tx = ADCBUF2 >> 8; //Envia Valor da fonte de Corrente
B2Tx = ADCBUF2 & 0x00FF;
CmdTx = 0x02;
break;
case 0x03:
Header = NS;
B1Tx = Speed >> 8; //Envia Valor da fonte da Velocidade
B2Tx = Speed & 0x00FF;
CmdTx = 0x03;
break;
}
TxData++;
while(BusyUART1()); //Retorna o status da transmissão; espera enquanto a UART estiver ocupada
putcUART1(Header); //Transmite via UART o Número de Série
putcUART1(B1Tx); //Transmite via UART o Parâmetro 1
putcUART1(B2Tx); //Transmite via UART o Parâmetro 2
putcUART1(CmdTx); //Transmite via UART o Comando
```

```
while(BusyUART1()); //Espera a UART desocupar
//1=UART ocupada, 0=UART desocupada
}
//*****************************************************************************************
//****************************** Fim ******************************************************
//*****************************************************************************************
******
```

APÊNDICE C- Código-fonte de malha fechada

Este apêndice contém o código-fonte desenvolvido para testes iniciais de malha fechada com Implementação de controle P.I.

```
/*
Autor: Leandro Borges - Universidade Feevale
Data: 20/11/2017
Titulo: Teste de Placa Protótipo com P.I.
Ver: V1.2 APÊNDICE C
Assunto:01 - LED1 de atividade na saida RC13
02 - Botão de Start/Stop na entrada RC14
03 - MCPWM
04 - OpenLoop nas saídas do PWM
05 - Start/Stop
06 - Controle de velocidade pelo potenciômentro
07 - Sensor Hall habilitado em CN5,6 and 7
08 - Verificação tensão de 24V
09 - Verificação de corrente
10 - Cálculo de Velocidade
11 - Transmissão via UART
12 - Controle P.I.
---------------------------------------------------------------------------
Mapeamento das ligações do DSPIC30F2010 aos outros módulos
---------------------------------------------------------------------------
Pino Função Pino Ligação dos pinos aos módulos utilizados
---------------------------------------------------------------------------
01 MCLR Reset e pino 6 para gravação in-circuit
02 EMUD3/AN0/VREF+/CN2/RB0 J4-1 Ch1 = AN0 = V24
03 EMUC3/AN1/VREF-/CN3/RB1 J4-2 Ch2 = AN1 = IMotor
04 AN2/SS1/CN4/RB2 J4-3 Ch0 = AN2 = Potenciômetro
05 AN3/INDX/CN5/RB3 J4-4 Hall A
06 AN4/QEA/IC7/CN6/RB4 J4-5 Hall B
07 AN5/QEB/IC8/CN7/RB5 J4-6 Hall C
08 VSS j4-7 e J6-1 Terra
09 OSC1/CLK1 Cristal 20MHz para clock
10 OSC2/CLKO/RC15 Cristal 20MHz para clock
11 EMUD1/SOSC1/T2CK/U1ATX/CN1/RC13 J6-8 - Led de atividade, 1K + Led + gnd
12 EMUC1/SOSC0/T1CK/U1ARX/CN0/RC14 J6-7 - Botão Start/Stop
13 VDD j4-8 VDD 5,00V
14 PREMUD2/OC2/IC2/INT2/RD1 J6-4 NC
15 EMUCS/OC1/IC1/INT1/RD0 J6-5 NC
16 FLTA/INT0/SCK1/OCFA/RE8 J6-6 - Jumper de Direção(NO) => "0" Rotacao Direita
17 PGD/EMUD/U1TX/SDO1/SDA/RF3 UART1 ligada ao Max 232 com jumper JP4
18 PGC/EMUC/U1RX/SDI1/SDA/RF2 UART1 ligada ao Max 232 com jumper JP3
19 VSS J6-1 e J4-7 Ground
```

```
20 VDD J4-8 VDD 5,00V
21 PWM3H/RE5 J5-1 PWM5 - Fase C Preto
22 PWM3L/RE4 J5-2 PWM4 - Fase C Preto
23 PWM2H/RE3 J5-3 PWM3 - Fase B Vermelho
24 PWM2L/RE2 J5-4 PWM2 - Fase B Vermelho
25 PWM1H/RE1 J5-5 PWM1 - Fase A Amarelo
26 PWM1L/RE0 J5-6 PWM0 - Fase A Amarelo
27 AVSS Terra virtual
28 AVDD VCC virtual
---------------------------------------------------------------------------
---------------------------------------------------------------------------
*/
//*************************************************************************************************
//***** Bibliotecas Utilizadas ********************************************************************
//*************************************************************************************************
#include<p30f2010.h> //Biblioteca do dsPIC30F2010
#include<uart.h> //Biblioteca UART.h
#include<pwm.h> //Biblioteca PWM.h
#include<reset.h> //Biblioteca reset.h
//*************************************************************************************************
//***** Fuses *************************************************************************************
//*************************************************************************************************
_FOSC(CSW_FSCM_OFF & HS); // Oscilador externo sem tratamento de erro
_FWDT(WDT_OFF); // WatchDog disabilitado
_FBORPOR(MCLR_EN & PBOR_OFF & PWRT_OFF); // Habilita MCLR, Desabilita PBOR e PWRT
_FGS(CODE_PROT_OFF); // Sem protecao de codigo - é possível a leitura
// posterior a gravação do chip
//*************************************************************************************************
//*************************************************************************************************
/*
Type Bits Min Max
char, signed char 8 -128 127
unsigned char 8 0 255
short, signed short 16 -32768 32767
unsigned short 16 0 65535
int, signed int 16 -32768 32767
unsigned int 16 0 65535
long, signed long 32 -2^31 2^(31 - 1)
unsigned long 32 0 232 - 1
long long**, signed long long** 64 -2^63 2^(63 - )1
unsigned long long** 64 0 2^(64 - 1)
Type Bits E Min E Max N Min N Max
float 32 -126 127 2^(-126) 2^128
double* 32 -126 127 2^(-126) 2^128
long double 64 -1022 1023 2^(-1022) 2^1024
*/
//*************************************************************************************************
//**** Define Valores Padrões para o Programa *****************************************************
```

```
//*************************************************************************************************
#define Fcy 5000000 //Freqüência do ciclo=freqüência do oscilador/4
//dsPIC necessita de 4 ciclos de clock para
// executar um ciclo de instrução
#define FPWM 16000 //Frequência MCPWM
#define NS 0x55 //Define a constante para ID de comunicacao
#define RPMConstant 60*(Fcy/256) //Define Constante 60*10000000/256 = 234375
// cálculo da velocidade
#define LED1 LATCbits.LATC13 //Define LED1 no pino RC13
#define S2 PORTCbits.RC14 //Define Botão Start/Stop no pino RC14
#define S3 PORTEbits.RE8 //Define Juper de direção da rotacao
#define V24 ADCBUF1 //Define V24 na porta RB0:
// verifica valor de tensão de 24V no circuito
#define IMotor ADCBUF2 //Define IMotor na porta RB0
// verifica valor da corrente através do sensor de corrente
//*************************************************************************************************
//******** Declarando as variáveis ***************************************************************
//*************************************************************************************************
unsigned int StateLoTable[] = //Sentido de rotação esquerda
{0x0000, 0x0210, 0x2004, 0x0204, 0x0801, 0x0810, 0x2001, 0x0000};
unsigned int StateRoTable[] = //Sentido de rotação direita
{0x0000, 0x2001, 0x0810, 0x0801, 0x0204, 0x2004, 0x0210, 0x0000};
unsigned int i; //Variável de controle de loops
unsigned int HallValue = 1; //Variável sensor hall
unsigned char RunMotor = 0; //Flag para start/stop 0 = stop
unsigned char Direction = 0; //Flag para sentido de rotacao 0 = esquerda
int Speed; //Variável de velocidade
unsigned int timer3value; //Variável Valor de Timer 3
unsigned int timer3avg; //Variável com a média do valor de Timer 3
unsigned char polecount; //Variável da contagem dos polos do motor
unsigned char TxData = 0; //Variável para envio do pacote de dados via uart
unsigned char S1 = 0; // Start/Stop via uart
unsigned char S4 = 1; // Sentido de rotacao via uart
int DesiredSpeed; //Variável da Velocidade Desejada
int ActualSpeed; //Variável da Velocidade Atual
int SpeedError; //Variável do Erro da Velocidade
int DutyCycle; //Variável de Correção do PID
int SpeedIntegral; //Váriavel do erro integral da Velocidade
#define POTMULT 10 //Multiplicador da Taxa de Velocidade
short int Kps = 4352; //0x1100 //100*Kp do Controle PI
short int Kis = 192; //0x00C0 //100*Ki do Controle PI
short int K=0; //Variável Para Mudança de Kp e Ki via Uart
//*********Protocolo do pacote de comunicação = Head, byte1, byte2, Comando **********************
unsigned char Header; //define a variável caracter Header
unsigned char B1Tx; //define a variável para o Byte 1 do pacote de transmissao
unsigned char B2Tx; //define a variável para o Byte 2 do pacote de transmissao
unsigned char B1Rx; //define a variável para o Byte 1 do pacote de recepcao
unsigned char B2Rx; //define a variável para o Byte 2 do pacote de recepcao
```

```
unsigned char CmdRx; //define a variável para o Comando a ser executado
unsigned char CmdTx; //define a variável para Comando a ser transmitido
//*****************************************************************************************
//************ Funções Utilizadas*********************************************************
//*****************************************************************************************
//*************************************************************************
void delayms(int d){ double i; int f; f = 13*d;
for (i=0;i<f;i++){} } // Para d = 1, atraso de tempo de 1 mS
void InitMCPWM(void); //Declara a função iniciar PWM
void InitADC10(void); //Declara a função iniciar ADC
void inicia_timer1(void); //Declara a função iniciar Timer1
void InitTMR3(void); //Declara a função iniciar Timer3
void __attribute__((interrupt, auto_psv))_ADCInterrupt(void); //Declara a função interrupção ADC
void __attribute__((interrupt, auto_psv)) _T1Interrupt( void ); //Declara a função interrupção T1
void inicia_UART(void); //Declara a função iniciar UART1
void __attribute__((interrupt, auto_psv))_U1TXInterrupt(void); //Interrupção de transmissão de dados
void __attribute__((interrupt, auto_psv))_U1RXInterrupt(void); //Interrupção de recepção de dados
void TransPac (void); //Declara a função para transmitir um pacote de 4 bytes via UART
void Start(void); //Declara a função Start, que retira o motor do repouso
void Stop(void); //Declara a função Stop, que coloca o motor em repouso
//*****************************************************************************************
//***** Rotina Principal do Programa *****////////////////////////////////////////////////////
//*****************************************************************************************
int main(void)
{
TRISD = 0xffff; //RD0 e RD1 p/ entrada. Não utilizada
TRISC = 0xdfff; //RC13 (LED1) p/ saída e RC14 (Botão) p/ entrada
TRISE = 0xFFC0; //RE0, 1, 2, 3, 4 e 5 p/ saída dos PWMs
CNEN1 = 0x00E0; //CN5,6 e 7 ativadas (Sensor Hall)
CNPU1 = 0x0000; //Desativa pullups internos do Processador
IFS0bits.CNIF = 0; //Limpa o registrador de interrupção (flag)
IEC0bits.CNIE = 1; //Ativa a interrupção CN
inicia_UART(); //Inicializa UART
inicia_timer1(); //Inicializa o Timer 1
InitTMR3(); //Inicializa o Timer 3
InitADC10(); //Inicializa a conversão A/D
InitMCPWM(); //Inicializa o MCPWM
while(1){ //Loop Infinito
while(S2) //Botão não pressionado
{
if( RunMotor == 1 && IMotor > 0x3FF)
{ //Se corrente maior que 5A
Stop(); //Para o motor e
for (i=0; i<10; i++) //Pisca o LED 10 vezes.
{LED1=!LED1; delayms(100);}
}
}
while(!S2) //Mudança de estado botão
```

```
{
if( RunMotor == 1) //Se o motor rodando
Stop(); //Desativa o motor
else //Se motor parado
{ //Se tensão entre 12V e 24V
if (V24 > 0x1FF && V24 < 0x3F0)
Start(); //Ativa o motor
else //Se não
{
for (i=0; i<5; i++) //Pisca o LED 5 vezes.
{
LED1=!LED1;
delayms(100);
}}}}}}
void Start(void)
{
while(S2) //Aguarda liberação do botão
{
HallValue = PORTB & 0x0038; //Mascara de bits em RB3,4 & 5
HallValue = HallValue >> 3; //Deslocamento de bits 3 vezes
Direction = S3; //Direção de rotação
if (Direction == 1) //Se Direção 1 Rotação para direita
OVDCON = StateRoTable[HallValue];//Carrega o registrador de sobreposição
else //Se direção 0, Rotação para esquerda
OVDCON = StateLoTable[HallValue];//Carrega o registrador de sobreposição
PDC1 = 0x500; //Inicia com PWM de valor elevado
PDC2 = PDC1; // devido a inércia.
PDC3 = PDC1;
PWMCON1 = 0x0777; //Habilita as saídas PWM
delayms(10); //Aguarda 10ms
LED1 = 1; //Ativa o LED1
RunMotor = 1; //Altera flag para motor rodando
goto fim; //Sai do laço While.
}
fim:;
}
void Stop(void)
{
while(S2) //Aguarda liberação do botão
{
PWMCON1 = 0x0700; //Desabilita as saidas PWM
OVDCON = 0x0000; //Desabilita o Registrador de sobreposição
delayms(10); //Aguarda 10ms
RunMotor = 0; //Altera flag para motor rodando
LED1 = 0; //Ativa o LED1
goto fim; //Sai do laço While.
}
fim:;
```

```
}
//*********************************************************************************************
//********************** Tratamento interrupção Timer 1 ***************************************
//*********************************************************************************************
void __attribute__((interrupt, auto_psv)) _T1Interrupt( void )
{
IFS0bits.T1IF = 0; //Reset timer interrupt flag
TransPac();
if( RunMotor == 0 && i==30 ) //Motor parado
{
LED1=!LED1; // LED pisca 5 vezes
i=0;
}
else i++;
}
void __attribute__((interrupt, auto_psv))_CNInterrupt(void)
{
IFS0bits.CNIF = 0; //Reseta a flag de interrupção
HallValue = PORTB & 0x0038; //Mascara em RB3,4 & 5
HallValue = HallValue >> 3; //Deslocamento para direita 3 vezes
if (Direction == 1)
OVDCON = StateRoTable[HallValue]; //Carrega registrador de sobrepor
else OVDCON = StateLoTable[HallValue]; //Carrega registrador de sobrepor
//Cálculo da velocidade.
if (HallValue == 1) //A mesma posição já foi verificada?
if (polecount++ == 2) //Dois pares de polos, uma volta
{
timer3value = TMR3; //Guarda o valor de Timer 3
TMR3 = 0; //Zera o Timer 3
timer3avg = ((timer3avg + timer3value) >> 1);//Calcula a média,
polecount = 1; //Reseta a contagem dos polos
Speed = RPMConstant/timer3avg; //Calcula a velocidade
}
}
//*********************************************************************************************
//************ Tratamento interrupção Conversor Analogico Digital *********************************
//*********************************************************************************************
void __attribute__((interrupt, auto_psv))_ADCInterrupt(void)
{
IFS0bits.ADIF = 0; //Leitura do Potenciomentro
DesiredSpeed = ADCBUF0/3;
CalculateDC();
}
//*********************************************************************************************
//************************* Rotina de inicialização do timer 1 ************************************
//*********************************************************************************************
void inicia_timer1(void)
{
```

```
T1CON = 1; //Desabilita a contagem do timer1
TMR1 = 0; //Zera a contagem de tempo do timer1
IFS0bits.T1IF = 0; //Clear no bit de chamada da interrupção do timer1
PR1 = Fcy/512;//128;//256; //Ajusta o período de tempo do timer1 para o valor da freqüência
// do ciclo de instrução/256 = 1 segundo
T1CONbits.TCKPS = 1; //Ajusta o prescaler
IPC0bits.T1IP = 7; //Define a prioridade da interrupção do timer1 = 7
IEC0bits.T1IE = 1; //Habilita a interrupção do timer1
T1CONbits.TON = 1; //1 = habilita a contagem do timer1
}
//*********************************************************************************************
//********************** Rotina de inicialização do timer 3 ***********************************
//*********************************************************************************************
void InitTMR3(void)
{
T3CON = 0x0030; // internal Tcy/256 clock
TMR3 = 0;
PR3 = 0xffff; //Fcy/256; //
T3CONbits.TON = 1; //1 = habilita a contagem do timer3
}
//*********************************************************************************************
//***** Inicialização do módulo ADC de 10 Bits ****************************************************
//*********************************************************************************************
void InitADC10(void)
{
ADPCFG = 0xFFF8; // PORTB = Digital exceção de RB0, RB1 e RB2 = analógico
ADCON1 = 0x0064; // Ativa o MCPWM nas saidas PWM 1100100
ADCON2 = 0x030C; // Amostragem simultanea dos 3 canais analogicos
ADCON3 = 0x0302; // Utilização do clock interno Tad = internal 2*Tcy
ADCHS = 0x0002; // Conecta RB2/AN2 no CH0
IFS0bits.ADIF = 0; // Limpa a flag de interrupção do AD
IEC0bits.ADIE = 1; // Habilita a interrupção do AD
ADCON1bits.ADON = 1;// Liga o AD.
}
//*********************************************************************************************
//***** Inicialização do módulo MCPWM *************************************************************
//*********************************************************************************************
void InitMCPWM(void)
{
PTPER = Fcy/FPWM - 1; //Base de tempo para o periodo do registrador
PWMCON1 = 0x0700; //Desativa os PWMs
OVDCON = 0x0000; //Permite o controle usando o registrador OVDCON
PDC1 = 100; //PWM Inicial do Motor, Antes da Leitura do POT
PDC2 = 100;
PDC3 = 100;
SEVTCMP = PTPER; //Habilita a interrupção especial do ADC
PWMCON2 = 0x0F00; //Habilita postscale de 16 vezes
PTCON = 0x8000; //Inicializa o PWM
```

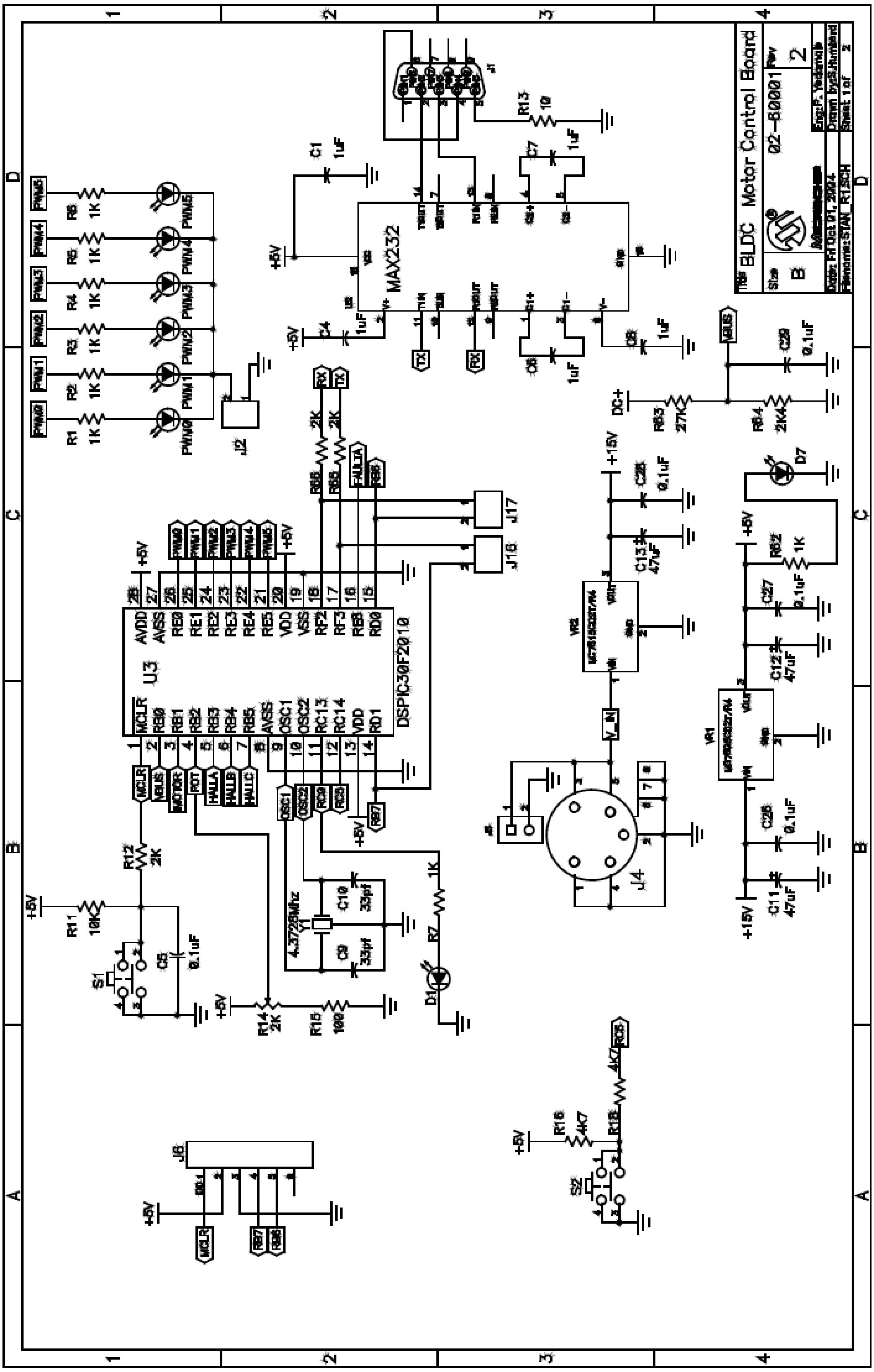
```
}
//*********************************************************************************************
//***** Inicialização do módulo UART ************************************************************
//*********************************************************************************************
void inicia_UART()
{
unsigned int ubrg; //Define a variável ubrg
unsigned int config1; //Define a variável config1
unsigned int config2; //Define a variável config2
CloseUART1(); //Desabilita a UART1
ubrg = 32; //Ubrg=(fcy/(BaudRate*16))-1
//Fcy=FCristal/4
//Neste Caso: Baud Rate=9600
//Freq.Cristal=20MHz
//Fcy=5MHz, ubrg=31,55
ConfigIntUART1(UART_RX_INT_EN & //Função para configuração das interrupções da UART
//Habilita a interrupção de recepção
UART_RX_INT_PR2 & //Define a prioridade da interrupção de recepção=2
UART_TX_INT_DIS & //Habilita a interrupção de transmissão
UART_TX_INT_PR4); //Define a prioridade da interrupção de transmissão=4
config1 = UART_EN & //Define a variável config1 com os seguintes parâmetros
//Habilita o módulo UART
UART_IDLE_CON & //UART funcionando no modo IDLE
UART_RX_TX & //Configura a comunicação através dos pinos padrões
// de TX e RX (U1TX e U1RX)
UART_DIS_WAKE & //Desabilita o modo "Wake-up"
UART_DIS_LOOPBACK & //Desabilita o modo "loop back"
UART_DIS_ABAUD & //Desabilita o modo "autobaud"
UART_NO_PAR_8BIT & //Define a comunicação com 8 bits de dados e
// sem bit de paridade
UART_1STOPBIT; //Define o Stop Bit=1
config2 = UART_INT_TX_BUF_EMPTY &
//Define a variável config2 com os seguintes parâmetros
//Ocorre interrupção de transmissão quando buffer de
// transmissão for esvaziado
UART_TX_PIN_NORMAL &//Define o bit TXBreak como normal
UART_TX_ENABLE & //Habilita a transmissão da UART
UART_INT_RX_BUF_FUL &
//Ativa a interrupção de recepção para Buffer cheio
UART_ADR_DETECT_DIS &
//Desativa o modo de "adress detect"
UART_RX_OVERRUN_CLEAR;
//Limpa o flag de "overrun"
OpenUART1(config1, config2, ubrg); //Habilita a UART1 com os parâmetros de configuração
// definidos pelas variáveis Config1, config2 e ubrg
}
//*********************************************************************************************
//********* Rotina de tratamento da Interrupção da recebimento de dados ***************************
```

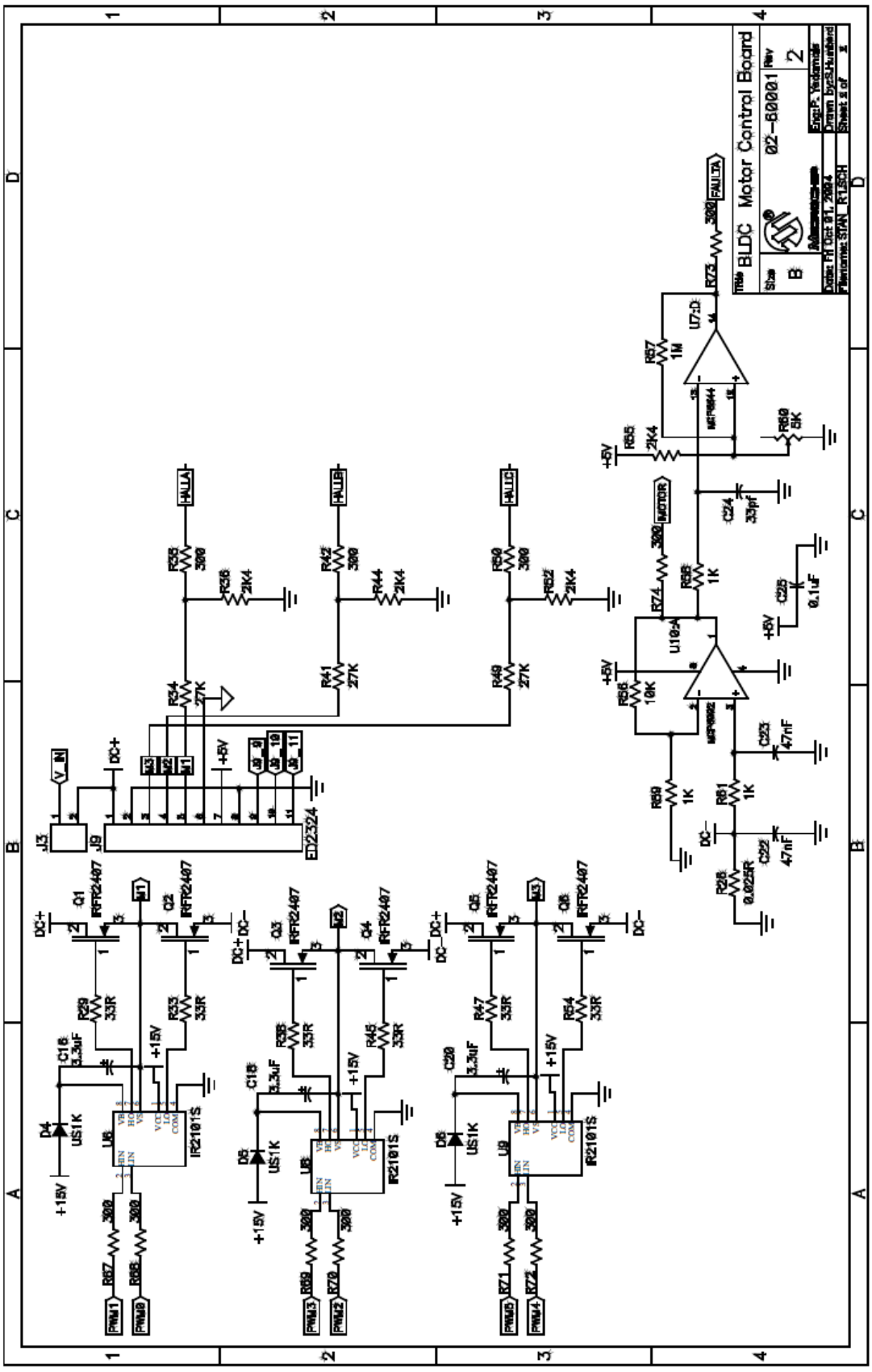
```
//*********************************************************************************************
void _ISR _U1RXInterrupt(void){ //Função da rotina de interrupção de recepção de dados
TMR1=0; //Zera a contagem do Timer 1
//Protocolo de comunicação = Head, byte1, byte2, Comando
while(BusyUART1()); //Retorna o status da transmissão;
//Espera enquanto a UART estiver ocupada
Header=ReadUART1(); //Lê o primeiro byte do buffer de recepção da UART = NS
B1Rx=ReadUART1(); //Lê o segundo byte do buffer de recepção da UART = Byte 1
B2Rx=ReadUART1(); //Lê o terceiro byte do buffer de recepção da UART = Byt 2
CmdRx=ReadUART1(); //Lê o quarto byte do buffer de recepção da UART = Comando
if(Header==NS){ //Compara se número de série recebido é igual ao correto(NS)
//Se for igual executa comando
K = B1Rx * 255 + B2Rx;
if (CmdRx==0){ Kps = K;} //Ajuste do KP Envio do comando 55 11 00 00 Kps = 4352
if (CmdRx==1) { Kis = K;} //Ajuste do KI Envio do comando 55 00 C0 01 Kis = 192
//Se for diferente ignora e retorna para a rotina principal
}
IFS0bits.U1RXIF=0; //Clear no bit de chamada da interrupção de recepção
} //Fim da interrupção de recepção da UART
//*********************************************************************************************
//***************Rotina para enviar o pacote de dados via UART ************************************
//*********************************************************************************************
void TransPac (void)
{ //Função que transmite o pacote de dados
switch(TxData & 0x03){//TxData & 0x03
case 0x00: //Caso Cmd=0x00 então executa os seguintes comandos
Header = NS;
B1Tx = ADCBUF0 >> 8; //Envia Valor do Potenciômetro
B2Tx = ADCBUF0 & 0x00FF;
CmdTx = 0x00;
break;
case 0x01:
Header = NS;
B1Tx = ADCBUF1 >> 8; //Envia Valor da fonte de Tensão
B2Tx = ADCBUF1 & 0x00FF;
CmdTx = 0x01;
break;
case 0x02:
Header = NS;
B1Tx = ADCBUF2 >> 8; //Envia Valor da fonte de Corrente
B2Tx = ADCBUF2 & 0x00FF;
CmdTx = 0x02;
break;
case 0x03:
Header = NS;
B1Tx = Speed >> 8; //Envia Valor da fonte da Velocidade
B2Tx = Speed & 0x00FF;
CmdTx = 0x03;
```

```
break;
}
TxData++;
while(BusyUART1()); //Retorna o status da transmissão; espera enquanto a UART estiver ocupada
putcUART1(Header); //Transmite via UART o Número de Série
putcUART1(B1Tx); //Transmite via UART o Parâmetro 1
putcUART1(B2Tx); //Transmite via UART o Parâmetro 2
putcUART1(CmdTx); //Transmite via UART o Comando
while(BusyUART1()); //Espera a UART desocupar
} //1=UART ocupada, 0=UART desocupada
void CalculateDC(void)
{
ActualSpeed = RPMConstant/timer3avg; //Velocidade atual
DesiredSpeed = DesiredSpeed*POTMULT; //Velocidade desejada
SpeedError = DesiredSpeed - ActualSpeed;//Diferença da Velocidade
SpeedIntegral += SpeedError; //Diferença do Erro da Velocidade
DutyCycle = //Fator de Correção PI
(((long)Kps/100*(long)SpeedError + (long)Kis/100*(long)SpeedIntegral) >> 16);
PDC1 = PDC1 + DutyCycle; //Alteração para PWM corrigido
if (PDC1 < 50)
{
PDC1 = 50; //Largura de pulso pequena quando
SpeedIntegral = 0; // pouco erro é detectado
}
if (PDC1 > 512)
{
PDC1 = 512; //Largura de pulso maior quando
SpeedIntegral = 0; // um erro maior é detectado
}
PDC2 = PDC1; //Largura de pulso corrigida
PDC3 = PDC1; // igualmente nas 3 fases
}
//*********************************************************************************************
//******************************** Fim ********************************************************
//*************************************************************************
```

ANEXO A- Esquema do controlador do motor

Este anexo contém diagramas esquemáticos para usar o dsPIC30F2010 para controlar um motor BLDC com sensor.


BLDC Motor Control Board
02-60001
Rev 2
Size B
Sheet 1 of 2
MAX232
U3
DSPIC30F2010
MCLR
RB0
RB1
RB2
RB3
RB4
RB5
AVSS
OSC1
OSC2
RC13
RC14
VDD
RD1
AVDD
RE0
RE1
RE2
RE3
RE4
RE5
VSS
RF2
RF3
RB8
RD0
PWM0
PWM1
PWM2
PWM3
PWM4
PWM5
R1
R2
R3
R4
R5
R6
1K
J2
C1
1uF
C4
C5
0.1uF
C6
C7
C8
R13
10
TX
RX
2K
FAULTA
J16
J17
+5V
+15V
DC+
R63
27K
R64
2K4
C29
C25
C13
47uF
C12
C11
C26
C27
D7
R62
VR1
VR2
V_IN
J4
R11
10K
R12
R14
R15
100
S1
C9
C10
33pf
4.3728Mhz
Y1
R7
D1
J6
R16
4K7
S2


BLDC Motor Control Board
02-60001
Rev 2
Sheet 1 of 1
Engr: P. Yedamale
Drawn by: Husfeldt
Date: Fri Oct 01, 2004
Filename: STAR_R1.SCH
Size B
Microchip
HALLA
HALLB
HALLC
MOTOR
FAULTA
V_IN
DC+
DC-
+15V
+5V
IR2101S
IRFR2407
US1K
MCP6044
MCP6002
DB22324

ANEXO B- Código-fonte de malha aberta

This appendix contains the source code listing for open-loop control.

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the "Company") is intended and supplied to you, the Company's customer, for use solely and exclusively with products manufactured by the Company.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

```
//------------------------------------------------------------------------------
//                      Software License Agreement
//
// The software supplied herewith by Microchip Technology Incorporated
// (the "Company") is intended and supplied to you, the Company's customer,
// for use solely and exclusively with products manufacture by the Company.
// The software is owned by the Company and/or its supplier, and is protected under
// applicable copyright laws. All rights are reserved. Any use in violation of the
// foregoing restrictions may subject the user to criminal sanctions under applicable
// laws, as well as to civil liability for the breach of the terms and conditions of
// this license.
//
// THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS,
// IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
// MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
// THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
// CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
//------------------------------------------------------------------------------
// File:  ClosedLoopSenBLDC.c
//
// Written By:Stan D'Souza, Microchip Technology
//
// The following files should be included in the MPLAB project:
//
//     ClosedLoopSenBLDC.c-- Main source code file
//     p30f2010.gld-- Linker script file
//
//
//------------------------------------------------------------------------
//
// Revision History
//
// 10/01/04 -- first version
//------------------------------------------------------------------------
/*************************************************************
    Low side driver table is as below.  In this StateLoTable,
    the Low side driver is PWM while the high side driver is
    either on or off.  This table is used in this exercise
*************************************************************/

unsigned int StateLoTable[] = {0x0000, 0x0210, 0x2004, 0x0204,
                               0x0801, 0x0810, 0x2001, 0x0000};
/*************************************************************
Interrupt vector for Change Notification CN5, 6 and 7 is as below.
When a Hall sensor changes states, an interrupt will be caused
which will vector to the routine below.
The user has to then read the PORTB, mask bits 3, 4 and 5,
shift and adjust the value to read as 1, 2 ... 6.  This
value is then used as an offset in the lookup table StateLoTable
to determine the value loaded in the OCDCON register
*************************************************************/
```

```
void _ISR _CNInterrupt(void)
{
    IFS0bits.CNIF = 0;                  // clear flag
    HallValue = PORTB & 0x0038;         // mask RB3,4 & 5
    HallValue = HallValue >> 3;         // shift right 3 times
    OVDCON = StateLoTable[HallValue];
}

/*************************************************************************
The ADC interrupt loads the PDCx registers with the demand pot value.
This is only done when the motor is running.

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

void _ISR _ADCInterrupt(void)
{
    IFS0bits.ADIF = 0;
    if (Flags.RunMotor)
    {
        PDC1 = ADCBUF0;                 // get value ...
        PDC2 = PDC1;                    // and load all three PWMs ...
        PDC3 = PDC1;                    // duty cycles
    }
}

int main(void)
{
    LATE = 0x0000;
    TRISE = 0xFFC0;                     // PWMs are outputs
    CNEN1 = 0x00E0;                     // CN5,6 and 7 enabled
    CNPU1 = 0x00E0;                     // enable internal pullups
    IFS0bits.CNIF = 0;                  // clear CNIF
    IEC0bits.CNIE = 1;                  // enable CN interrupt
    InitMCPWM();
    InitADC10();
    while(1)
    {       while (!S2);                // wait for start key hit
        while (S2)                      // wait till key is released
            DelayNmSec(10);
        // read hall position sensors on PORTB
        HallValue = PORTB & 0x0038;     // mask RB3,4 & 5
        HallValue = HallValue >> 3;     // shift right to get value 1, 2 ... 6
        OVDCON = StateLoTable[HallValue]; // Load the overide control register
        PWMCON1 = 0x0777;               // enable PWM outputs
        Flags.RunMotor = 1;             // set flag
        while (Flags.RunMotor)          // while motor is running
            if (S2)                     // if S2 is pressed
            {
                PWMCON1 = 0x0700;       // disable PWM outputs
                OVDCON = 0x0000;        // overide PWM low.
                Flags.RunMotor = 0;     // reset run flag
                while (S2)              // wait for key release
                    DelayNmSec(10);
            }
    }   // end of while (1)


}
```

```
/*********************************************************************
       Below is the code required to setup the ADC registers for :
       1. 1 channel conversion (in this case RB2/AN2)
       2. PWM trigger starts conversion
       3. Pot is connected to CH0 and RB2
       4. Manual Stop Sampling and start converting
       5. Manual check of Conversion complete

*********************************************************************/
void InitADC10(void)
{

 ADPCFG = 0xFFF8;                  // all PORTB = Digital;RB0 to RB2 = analog
 ADCON1 = 0x0064;                  // PWM starts conversion
 ADCON2 =  0x0200;                 // simulataneous sample 4 channels
 ADCHS = 0x0002;                   // Connect RB2/AN2 as CH0 = pot  ..
                                   // ch1 = Vbus, Ch2 = Motor, Ch3 = pot
 ADCON3 = 0x0080;                  // Tad = internal RC (4uS)
 IFS0bits.ADIF = 0;
 IEC0bits.ADIE = 1;

 ADCON1bits.ADON = 1;              // turn ADC ON
}

/*********************************************************************
InitMCPWM, intializes the PWM as follows:
1. FPWM = 16000 hz
2. Independant PWMs
3. Control outputs using OVDCON
4.  Set Duty Cycle with the ADC value read from pot
5. Set ADC to be triggered by PWM special trigger
*********************************************************************/

void InitMCPWM(void)
{
    PTPER = FCY/FPWM - 1;

    PWMCON1 = 0x0700;              // disable PWMs
    OVDCON = 0x0000;               // allow control using OVD
    PDC1 = 100;                    // init PWM 1, 2 and 3 to 100
    PDC2 = 100;
    PDC3 = 100;
    SEVTCMP = PTPER;
    PWMCON2 = 0x0F00;              // 16 postscale values
    PTCON = 0x8000;                // start PWM

}

//--------------------------------------------------------------------
// This is a generic 1ms delay routine to give a 1mS to 65.5 Seconds delay
// For N = 1 the delay is 1 mS, for N = 65535 the delay is 65,535 mS.
// Note that FCY is used in the computation.  Please make the necessary
// Changes(PLLx4 or PLLx8 etc) to compute the right FCY as in the define
// statement above.

void DelayNmSec(unsigned int N)
{
unsigned int j;
while(N--)
    for(j=0;j < MILLISEC;j++);
}
```

ANEXO C- Código-fonte de malha fechada

```
//------------------------------------------------------------------------------
//                      Software License Agreement
//
// The software supplied herewith by Microchip Technology Incorporated
// (the "Company") is intended and supplied to you, the Company's customer,
// for use solely and exclusively with products manufacture by the Company.
// The software is owned by the Company and/or its supplier, and is protected under
// applicable copyright laws. All rights are reserved. Any use in violation of the
// foregoing restrictions may subject the user to criminal sanctions under applicable
// laws, as well as to civil liability for the breach of the terms and conditions of
// this icense.
//
// THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES, WHETHER EXPRESS,
// IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
// MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
// THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
// CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
//------------------------------------------------------------------------------
// File:  ClosedLoopSenBLDC.c
//
// Written By:Stan D'Souza, Microchip Technology
//
// The following files should be included in the MPLAB project:
//
//     ClosedLoopSenBLDC.c-- Main source code file
//     p30f2010.gld-- Linker script file
//
//------------------------------------------------------------------------
// Revision History
//
// 10/01/04 -- first version
//------------------------------------------------------------------------
//************************************************************************
ClosedLoopSenBLDC.c is a sensored Closed Loop Control for a BLDC motor.
The task consists of the following:
Sense changes in the Hall Sensors connected to CN5,6 & 7 (PortB)
During the CNInterrupt, read the sensors input by reading PortB
Mask and determine the state of the position 1, 2, ... 6.
Use the StateLoTable and the lookup table provided to determine the
Overload Control Register value.  Set the OVDCON to this value.

The PWM is initialized to generate independant continuous PWMs.
The value of the Pot REF is used to determine the demand or desired
speed of the Motor.  The desired speed value is then used with the
actual speed value to determine the Proportional Speed Error and the
Integral Speed Error.  With these two values the new DutyCycle is determined
as: NewDutyCycle = Kp*(Portportional Speed Error) + Ki*(Integral Speed Error)
All 3 PWM Duty cycles are then loaded with the NewDutyCycloe 10-bit value.

The FPWM = 16000hz

The ADC is setup for a PWM trigger to start the conversion
************************************************************************/
```

```
#define __dsPIC30F2010__
#include "c:\pic30_tools\support\h\p30F2010.h"


#define FCY  10000000// xtal = 5.0Mhz; PLLx8
#define MILLISEC FCY/10000// 1 mSec delay constant
#define FPWM 16000
#define Ksp1200
#define Ksi10
#define RPMConstant60*(FCY/256)

#define S2!PORTCbits.RC14


void InitTMR3(void);
void InitADC10(void);
void AverageADC(void);
void DelayNmSec(unsigned int N);
void InitMCPWM(void);
void CalculateDC(void);
void GetSpeed(void);

struct {
          unsigned RunMotor : 1;
          unsigned Minus : 1;
          unsigned unused : 14;
       }  Flags;

unsigned int HallValue;
int Speed;
unsigned int Timer3;
unsigned char Count;
unsigned char SpeedCount;
int DesiredSpeed;
int ActualSpeed;
int SpeedError;
int DutyCycle;
int SpeedIntegral;

//*************************************************************
   Low side driver table is as below.  In this StateLoTable,
   the Low side driver is PWM while the high side driver is
   either on or off.  This table is used in this exercise
*************************************************************/

unsigned int StateLoTable[] = {0x0000, 0x1002, 0x0420, 0x0402,
                               0x0108, 0x1008, 0x0120, 0x0000};
/*************************************************************
Interrupt vector for Change Notification CN5, 6 and 7 is as below.
When a Hall sensor changes states, an interrupt will be caused which
will vector to the routine below.
The user has to then read the PORTB, mask bits 3, 4 and 5,
shift and adjust the value to read as 1, 2 ... 6.  This
value is then used as an offset in the lookup table StateLoTable
to determine the value loaded in the OCDCON register
*************************************************************/
```

```
void _ISR _CNInterrupt(void)
{
    IFS0bits.CNIF = 0;                  // clear flag
    HallValue = PORTB & 0x0038;         // mask RB3,4 & 5
    HallValue = HallValue >> 3;         // shift right 3 times
    OVDCON = StateLoTable[HallValue];// Load the overide control register
}

/*************************************************************************
The ADC interrupt loads the DesiredSpeed variable with the demand pot
value.  This is then used to determing the Speed error.  When the motor
is not running, the PDC values use the direct Demand value from the pot.
*************************************************************************/
void _ISR _ADCInterrupt(void)
{
    IFS0bits.ADIF = 0;
    DesiredSpeed = ADCBUF0;
    if (!Flags.RunMotor)
        {
        PDC1 = ADCBUF0;                 // get value ...
        PDC2 = PDC1;                    // and load all three PWMs ...
        PDC3 = PDC1;                    // duty cycles
        }
}

/*************************************************************************
The main routine controls the initialization, and the keypress to start
and stop the motor.
*************************************************************************/

int main(void)
{
    LATE = 0x0000;
    TRISE = 0xFFC0;                 // PWMs are outputs
    CNEN1 = 0x00E0;                 // CN5,6 and 7 enabled
    CNPU1 = 0x00E0;                 // enable internal pullups
    IFS0bits.CNIF = 0;              // clear CNIF
    IEC0bits.CNIE = 1;              // enable CN interrupt
    SpeedError = 0;
    SpeedIntegral = 0;
    InitTMR3();
    InitMCPWM();
    InitADC10();
    while(1)
    {
        while (!S2);                // wait for start key hit
        while (S2)                  // wait till key is released
           DelayNmSec(10);
        // read hall position sensors on PORTB
        HallValue = PORTB & 0x0038;         // mask RB3,4 & 5
        HallValue = HallValue >> 3;         // shift right to get value 1, 2 ... 6
        OVDCON = StateLoTable[HallValue];// Load the overide control register
        PWMCON1 = 0x0777;                   // enable PWM outputs
        Flags.RunMotor = 1;                 // set flag
        T3CON = 0x8030;                     // start TMR3
        while (Flags.RunMotor)              // while motor is running
           if (!S2)                         // if S2 is not pressed
```

```
            {
               if (HallValue == 1)        //IF in sector 1
               {
               HallValue = 0xFF;             // force a new value as a sector
               if (++Count == 5)             // do this for 5 electrical revolutions or 1
                                             // mechanical revolution for a 10 pole motor
                  {
                  Timer3 = TMR3;// read latest tmr3 value
                  TMR3 = 0;
                  Count = 0;
                  GetSpeed();// determine spped
                  }
               }
            }
            else          // else S2 is pressed to stop motor
            {
               PWMCON1 = 0x0700;// disable PWM outputs
               OVDCON = 0x0000; // overide PWM low.
               Flags.RunMotor = 0;// reset run flag
               while (S2)// wait for key release
                  DelayNmSec(10);
            }
      }  // end of while (1)

}

/*********************************************************************
      Below is the code required to setup the ADC registers for :
      1. 1 channel conversion (in this case RB2/AN2)
      2. PWM trigger starts conversion
      3. Pot is connected to CH0 and RB2
      4. Manual Stop Sampling and start converting
      5. Manual check of Conversion complete

*********************************************************************/
void InitADC10(void)
{

 ADPCFG = 0xFFF8;                 // all PORTB = Digital;RB0 to RB2 = analog
 ADCON1 = 0x0064;                 // PWM starts conversion
 ADCON2 =  0x0000;                // sample CH0 channel
 ADCHS = 0x0002;                  // Connect RB2/AN2 as CH0 = pot.
 ADCON3 = 0x0080;                 // Tad = internal RC (4uS)
 IFS0bits.ADIF = 0;               // clear flag
 IEC0bits.ADIE = 1;               // enable interrupt

 ADCON1bits.ADON = 1;             // turn ADC ON
}
```

```
/*********************************************************************
InitMCPWM, intializes the PWM as follows:
1. FPWM = 16000 hz
2. Independant PWMs
3. Control outputs using OVDCON
4. Set Duty Cycle using PI algorithm and Speed Error
5. Set ADC to be triggered by PWM special trigger
*********************************************************************/

void InitMCPWM(void)
{
    PTPER = FCY/FPWM - 1;

    PWMCON1 = 0x0700;          // disable PWMs
    OVDCON = 0x0000;           // allow control using OVD
    PDC1 = 100;                // init PWM 1, 2 and 3 to 100
    PDC2 = 100;
    PDC3 = 100;
    SEVTCMP = PTPER;           // special trigger is 16 period values
    PWMCON2 = 0x0F00;          // 16 postscale values
    PTCON = 0x8000;            // start PWM

}

/*********************************************************************
Tmr3 is used to determine the speed so it is set to count using Tcy/256

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

void InitTMR3(void)
{
    T3CON = 0x0030;            // internal Tcy/256 clock
    TMR3 = 0;
    PR3 = 0x8000;
}

/*********************************************************************
GetSpeed, determins the exact speed of the motor by using the value in
TMR3 for every mechanical cycle.
*********************************************************************/

void GetSpeed(void)
{
    if (Timer3 > 23000)                 // if TMR3 is large ignore reading
        return;
    if (Timer3 > 0)
        Speed = RPMConstant/(long)Timer3;// get speed in RPM
    ActualSpeed += Speed;
    ActualSpeed = ActualSpeed >> 1;
    if (++SpeedCount == 1)
        {SpeedCount = 0;CalculateDC();}
}
```

```
/*************************************************************************
CalculateDC, uses the PI algorithm to calculate the new DutyCycle value which
will get loaded into the PDCx registers.
*************************************************************************/

void CalculateDC(void)
{

    DesiredSpeed = DesiredSpeed*3;
    Flags.Minus = 0;
    if (ActualSpeed > DesiredSpeed)
    SpeedError = ActualSpeed - DesiredSpeed;
    else
        {
        SpeedError = DesiredSpeed - ActualSpeed;
        Flags.Minus = 1;
        }
    SpeedIntegral += SpeedError;
    if (SpeedIntegral > 9000)
        SpeedIntegral = 0;
    DutyCycle = (((long)Ksp*(long)SpeedError + (long)Ksi*(long)SpeedIntegral) >> 12);
    DesiredSpeed = DesiredSpeed/3;
    if (Flags.Minus)
    DutyCycle = DesiredSpeed + DutyCycle;
    else DutyCycle = DesiredSpeed - DutyCycle;
    if (DutyCycle < 100)
        DutyCycle = 100;
    if (DutyCycle > 1250)
        {DutyCycle = 1250;SpeedIntegral = 0;}
    PDC1 = DutyCycle;
    PDC2 = PDC1;
    PDC3 = PDC1;

}

//-----------------------------------------------------------------------
// This is a generic 1ms delay routine to give a 1mS to 65.5 Seconds delay
// For N = 1 the delay is 1 mS, for N = 65535 the delay is 65,535 mS.
// Note that FCY is used in the computation.  Please make the necessary
// Changes(PLLx4 or PLLx8 etc) to compute the right FCY as in the define
// statement above.

void DelayNmSec(unsigned int N)
{
unsigned int j;
while(N--)
    for(j=0;j < MILLISEC;j++);
}
```