Faculdade de Engenharia da Universidade do Porto

Sistema de 'drive' de múltiplos motores brushless para utilização em robótica móvel

Rui Fernando de Sousa Marques

VERSÃO FINAL

Dissertação realizada no âmbito do
Mestrado Integrado em Engenharia Electrotécnica e de Computadores
Major Automação

Orientador: Armando Jorge Miranda de Sousa (Prof. Doutor)

Fevereiro de 2013

U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MIEEC - MESTRADO INTEGRADO EM ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES 2012/2013

A Dissertação intitulada

"Sistema de "Drive" de Múltiplos Motores Brushless para Utilização em Robótica Móvel"

foi aprovada em provas realizadas em 05-03-2013

o júri

Presidente Professor Doutor Armando Luís Sousa Araújo
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

Professor Doutor José Luís Magalhães Lima
Professor Adjunto Departamento de Eletrotécnia da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Bragança

Professor Doutor Armando Jorge Miranda de Sousa
Professor Auxiliar do Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projeto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extratos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são corretamente citados.

Autor - Rui Fernando de Sousa Marques

Faculdade de Engenharia da Universidade do Porto

Resumo

O objetivo principal desta dissertação consiste na implementação de um controlador para múltiplos motores brushless para aplicação em robótica móvel.

Procurando-se por uma certa flexibilidade no tipo de programação, foi escolhido um controlador a utilizar - FPGA, assim como a sua respetiva linguagem de programação - Verilog. De seguida foi realizada uma análise da estrutura de programação para realização do controlo de um robot de duas rodas. Este controlo deverá ser também capaz de privilegiar a correção do erro da velocidade angular sobre a velocidade linear do robot. A este controlo foi chamado "controlo de Velocidade Linear condicionada".

Para a implementação do controlo optou-se por uma interface em Lazarus. Esta é capaz de monitorizar o valor das velocidades linear e angular do robot, de cada motor e os respetivos consumos (correntes dos motores).

Na fase de implementação é descrito o modelo do motor utilizado, assim como o driver de potência que o alimenta. Neste capítulo serão também apresentadas simulações, resultados experimentais, assim como algumas dificuldades deparadas ao longo desta dissertação.

No último capítulo, serão realizadas as conclusões e propostas para continuação de desenvolvimento desta plataforma.

Abstract

The main objective of this dissertation is the implementation of a controller for multiples brushless DC motor for application in mobile robotics.

Looking for some flexibility in the programming, it was chosen the controller to be used - FPGA, as well as the programming language - Verilog. Then it was performed a programming structure analysis to be applied in a two-wheeled robot. The control of the robot should also be able to give priority to the correction of the angular speed, over the linear speed of the robot. This control was named "Linear Velocity Conditioned Control".

In order to control the robot, a Lazarus interface was created. This Lazarus interface is also able to monitor the linear and angular robot speed, the motors speed and also the motors current consumption.

In the implementation stage the motor model is presented as well as the power driver used to supply it. In this chapter it is also presented all the simulations and experimental results, as well as, some of the difficulties that occurred during this dissertation.

In the last chapter, conclusions are stated and future work related to the development of this platform is presented.

Agradecimentos

Em primeiro lugar gostaria de agradecer à Faculdade de Engenharia da Universidade do Porto por me proporcionar a oportunidade de concretização da presente dissertação, disponibilizando todos os recursos necessários.

Ao meu orientador, Professor Doutor Armando Jorge Miranda de Sousa, pela disponibilidade, interesse e apoio manifestados ao longo desta dissertação que me ajudaram no desenvolvimento do meu trabalho.

Ao professor José Carlos dos Santos Alves, pela disponibilização do espaço e material de laboratório, assim como esclarecimento de algumas questões de programação em Verilog.

Quero também agradecer aos meus colegas da sala I224, Nuno Paulino e Hélder Campos pela prontificação em ajudar a superar algumas dificuldades deparadas no desenvolvimento do meu trabalho.

À minha família, pela motivação concedida e me terem ajudado a superar algumas dificuldades. Em especial ao meu pai que sempre me motivou a dar "um pouco mais".

A todos os meus amigos pessoais, que sempre me incentivaram na concretização desta dissertação, em especial ao Cláudio Pinto, João Ramos, Nuno Lima, Rodolfo Marques, Rui Amaro e Tiago Raúl.

Por fim, quero ainda agradecer ao Rui Melo pelas longas conversas e caminhadas, à Ana Hierro pela sua constante presença e animação, e à Diana Oliveira que, apesar de estar longe, também conseguiu estar perto.

"…onde estão os outros dois pontos?"

F. Marques

Índice

Lista de figuras

Lista de tabelas

Abreviaturas e Símbolos

Lista de abreviaturas

AC	*Alternate Current* / corrente alternada
BJT	*Bipolar Junction Transistor* / Transístor de Junção Bipolar
BLDC	*Brushless DC (motor)* / Motor DC sem escovas
CLB	*Configurable Logic Block*
DC	*Direct Current* / corrente contínua
DEEC	Departamento de Engenharia Electrotécnica e de Computadores
DSP	Digital Signal Processor
EM	Electromagnetic
FEUP	Faculdade de Engenharia da Universidade do Porto
FPGA	*Field-Programmable Gate Array*
Fuzzy	*Fuzzy control*
GTO	*Gate Turn-off Thyristor*
HDL	*Hardware Description Language*
IGBT	*Insulated-Gate Bipolar Transistor*
IOB	*Input Output Blocks*
IM	*Induction Motor*
LUT	*Look Up Table*
MAC	*Multiply Accumulator*
MOSFET	*Metal-Oxide-Semiconductor Field-Effect Transistor*
PI	*Proportional + Integrator controller*
PLD	*Programmable Logic Device*
PMSM	*Permanent Magnet Synchronous Motor*
PWM	*Pulse Width Modulation*
SVM	*Space Vector Modulation*
VHDL	VHSIC Hardware Description Language
VHSIC	Very-High-Speed Integrated Circuits

Lista de símbolos

α	Ângulo
V_d, V_q	Tensões de eixo direto e eixo em quadratura
θ	Posição angular
ω	Velocidade angular
i_d,i_q	Correntes de eixo direto e eixo em quadratura
J	Momento de Inércia
p	Número de pares de pólos
L_d, L_q	Indutância de eixo direto e eixo em quadratura
R_s	Resistência de estator
K_m	Constante do fluxo do rotor de ímanes permanentes
P_m	Potência mecânica
P_{ma}	Potência referente à energia armazenada no campo magnético de eixo direto e em quadratura do estator

Capítulo 1

Introdução

Neste capítulo será realizada a contextualização do tema da dissertação, assim como a exposição dos motivos que levaram ao levantamento do tema da mesma e respetivos objetivos. No final, será explicitada a estruturação deste documento.

1.1 Contextualização

A aplicação de motores Brushless no quotidiano do dia-a-dia tem vindo a aumentar substancialmente com o progresso dos anos. Tanto em áreas de transportes (ex: Segway Scooter, Vectrix Maxi-Scooter), refrigeração (ex: HVAC de baixo consumo), indústria (ex: maquinação e transporte de peças), assim como em outros sistemas de controlo e posicionamento devido ao seu curto espaço de tempo de resposta à alteração de binário. Por esses e outros, devido à sua precisão e fiabilidade, sistemas antigos têm vindo a ser substituídos por sistemas baseados em motores BLDC. No entanto, apesar das vantagens dos motores BLDC, quando comparado com outros sistemas com outros tipos de motor, o seu método de controlo é complexo e elaborado, existindo uma grande variedade de abordagens possíveis para o seu controlo.

Também no âmbito da robótica móvel, outra tecnologia que tem vindo a distinguir-se e a integrar-se cada vez mais em sistemas autónomos consiste no desenvolvimento de plataformas PLD, mais concretamente as FPGAs. A sua capacidade de processamento distingue-se pela capacidade de executar funções em paralelo, aumentando assim a sua rapidez de cálculo e resposta.

Com intuito do estudo do controlo aplicado a um robot constituído por múltiplos motores BLDC, assim como do controlador acoplado a estes, surge a motivação desta dissertação.

1.2 Motivação e Objetivos

A capacidade de rápido processamento da FPGA, assim como da sua linguagem flexível e estruturada foi um forte motivador para a aplicação da mesma no controlo de motores BLDC (por intermédio de controlo de drivers de potência).

A sua capacidade de realização de um controlo do robot em concordância com o controlo dos drivers dos motores brushless, permite-nos aplicar os sinais adquiridos dos motores ao controlo do robot. Para além deste controlo dependente, a flexibilidade na programação das PLD, permite-nos abranger uma grande variedade de casos em robótica móvel, permitindo-nos alterar não só o método de controlo do robot, como dos motores.

Realizando o controlo da estrutura robótica e dos drivers dos motores num único controlador, a redução do volume e custo é evidente.

Na figura 1.1 podemos observar um diagrama do sistema geral implementado.

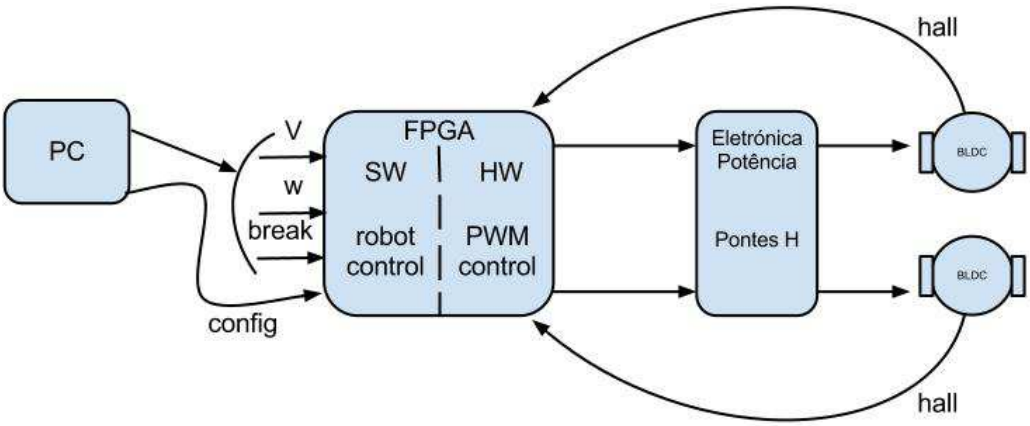


Figura 1.1 – Estrutura geral do programa

Por análise da figura 1.1 podemos observar que o comando da estrutura robótica será realizado num computador. Uma configuração deverá ser enviada previamente para a FPGA, permitindo depois ao utilizador um completo controlo sobre o robot.

Internamente à FPGA, o controlo das velocidades dos motores BLDC deverá ser realizado paralelamente ao comando enviado ao robot. O controlo do robot deverá receber as referências de velocidade linear (V_{robot}) e angular (ω_{robot}), realizando uma lógica definida pelo utilizador, apresentando à saída os valores das velocidades de referência de cada motor. Estes valores apresentados pelo bloco de controlo do robot serão depois aplicados ao controlo das velocidades dos motores que se encarregarão de produzir as PWM a aplicar nos drivers de potência que alimentam os motores BLDC.

Na falha de comunicação entre o computador e a FPGA, o sinal “break”, representado na figura 1.1, provocará uma paragem de emergência.

1.3 Estrutura da dissertação

Esta dissertação encontra-se dividida em 6 capítulos. No primeiro capítulo é realizada a introdução à temática discutida nesta dissertação, abordando-se também a motivação que levou à sua concretização.

No segundo capítulo é realizado o levantamento do estado de arte dos motores e conversores tipicamente utilizados em robótica móvel, dando especial ênfase ao controlo de motores imãs permanentes (BLDC). O estado da arte abordará também controladores PLD, em especial as FPGA.

No terceiro capítulo será realizada uma análise da PLD escolhida, sendo também explicada toda a programação implementada no controlador FPGA em Verilog.

No quarto capítulo será apresentada a aplicação gráfica desenvolvida em ambiente Lazarus para interação com o controlador FPGA e comando do robot, assim como todas as funcionalidades desta aplicação.

No quinto capítulo serão apresentados alguns resultados experimentais respetivos à aplicação dos sinais produzidos pela FPGA numa ponte em H, sendo também apresentadas algumas dificuldades deparadas.

A dissertação encerra no sexto capítulo, onde são realizadas as conclusões desta dissertação e expostas algumas propostas para desenvolvimentos futuros desta dissertação.

Capítulo 2

Estado da Arte

Neste capítulo será apresentado o estado da arte de controladores em PLD, e dos motores (e respetivos drivers de potência) mais utilizados na indústria da robótica móvel. Foi dado especial ênfase aos PLDs – FPGAs e aos motores de imãs permanente, BLDC.

2.1 Programmable Logic Devices (PLDs) – Perspetiva Histórica

Um PLD é um circuito integrado que permite o utilizador, através de uma devida configuração, a implementação de funções lógicas variadas de complexidade e extensão variada. Os PLDs podem ser classificados em várias categorias: [1]

1. Simple Programmable Logic Devices (SPLD)
    a. Programmable Logic Array (PLA): Um Array lógico programável é um circuito integrado que contém dois níveis de lógica de programação (planos AND e OR)
    b. Programmable Array Logic (PAL): Um PAL é um circuito integrado que contém um plano fixo OR, seguido de um plano programável AND
2. Complex Programmable Logic Device (CPLD)
3. Field Programmable Gate Array (FPGA)

2.2 FPGA – "Field Programmable Gate Array"

A sigla FPGA é um acrónimo de "Field Programmable Gate Array", e corresponde a um circuito integrado configurado para implementação de funções lógicas digitais de várias complexidades. A grande distinção das FPGAs face aos restantes PLDs (como por exemplo um chip DSP, que é limitado em número de Muliplier ACcumulator – MAC – disponíveis para o

design do controlador), consiste na sua capacidade de implementar um controlador digital em paralelo, resultando em velocidades de operação muito rápidas [1].

2.2.1 Arquitetura

Uma FPGA é tipicamente constituída pelas seguintes partes:

1. Programmable Logic blocks
2. Interconnection Resources
3. Input output blocks

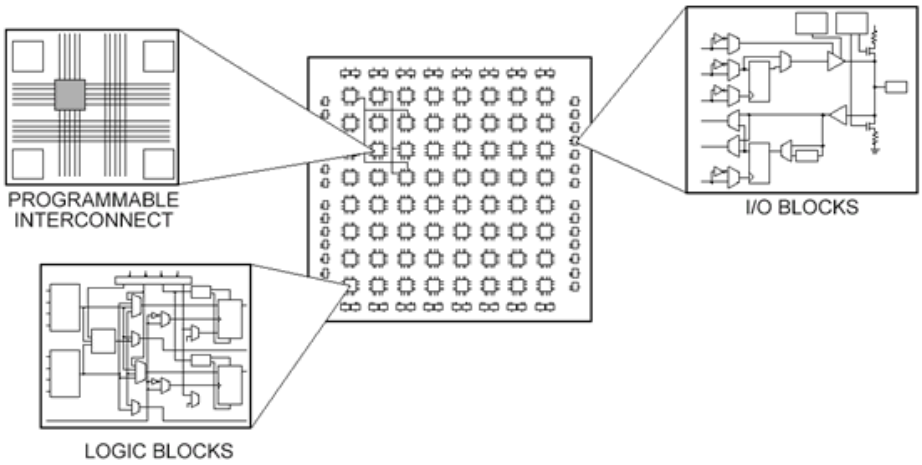


Figura 2.1 – Interior/constituição de uma FPGA [2].

2.2.1.1 Programmable Logic block

Este bloco é tipicamente constituído por Configurable Logic Blocks (CLBs). Estes CLBs podem ser implementados de várias maneiras, sendo no entanto o método mais comum o de “Look Up Tables” (LUT). Através destas, o utilizador pode implementar qualquer função de k variáveis de entrada, configurando tabelas de verdade das LUTs [3].

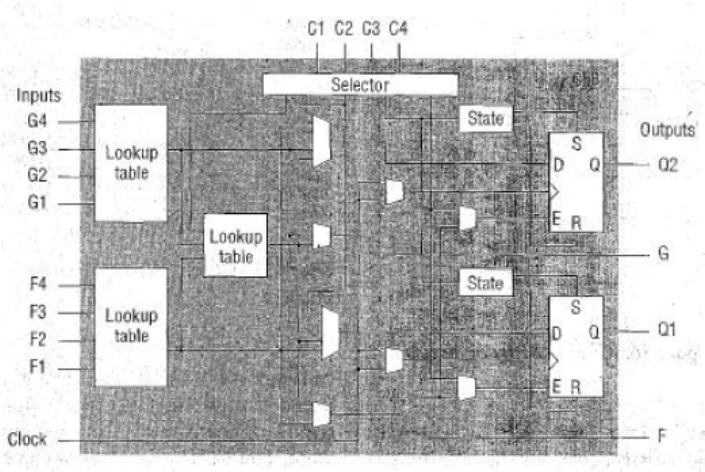


Figura 2.2 – CLB de 9 entradas [3].

Na figura 2.2 podemos observar um CLB capaz de implementar funções lógicas de 9 entradas. É possível configurar as LUT nas CLB para ler/escrever em posições específicas da RAM. Algumas das FPGAs mais modernas possuem micro controladores mais completos em chip, facilitando a implementação de expressões lógicas mais complexas [1].

2.2.1.2 Interconnection Resources

Outros recursos importantes constituintes da FPGA e que influenciam a sua performance, são as suas "Interconnection Resources". São através destas que os vários subsistemas que são implementados em diferentes CLBs, comunicam entre si, permitindo a implementação de um sistema completo e concentrado [1].

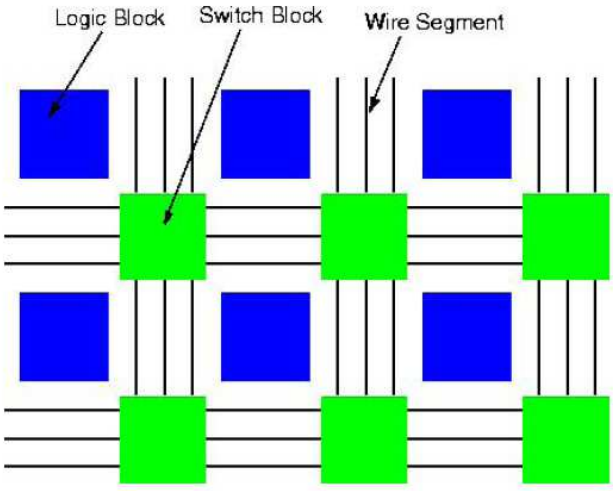


Figura 2.3 – Elementos internos à FPGA – Logic Block, Switch Block Wire Segment [1].

2.2.1.3 Input Output Blocks (IOBs)

Os IOBs proporcionam a interface entre a FPGA e os sinais adquiridos no sistema externo (“mundo real”). Estes blocos são constituídos por ilhas de ligação I/O, estando estes ligados aos pinos do IC, podendo os sinais exteriores ser declarados como “input para” ou “output de” um array de logic cells. Nestes blocos estão também presentes buffers, Flip-flops. Os IOBs estão também tipicamente equipados com resistências internas de “pull up” com objetivo de proteção ou evitar uma entrada float [4].

As FPGAs podem variar também na sua granularidade, desde dispositivos com um grande número de pequenos recursos de lógica programáveis (ex. Atmel AT40K) ou com um número inferior de recursos com maior complexidade como os LUTs e flip-flops (ex. Xilinx Virtex series) [1].

2.2.2 Comparação: FPGA vs DSP

A primeira pergunta que se deve colocar é: “porque utilizar controlo integrado através de uma FPGA?”. A resposta surge por análise das seguintes vantagens da FPGA. A maioria dos cálculos envolve o uso de 2 operadores, sendo o primeiro o operador multiplicativo e o segundo o operador acumulativo. Juntos, estes operadores são conhecidos como Multiply ACcumulate (MAC) operations. De uma forma geral, os recursos de um microprocessador são colocado em “stand by” enquanto estiver “busy” a realizar estas MAC operations, e a velocidade ou o tempo de amostragem é condicionado pela latência destas instruções [1].

Será realizada uma comparação da implementação de um filtro por uma FPGA e por um DSP (muitos destes controladores são implementados como filtros) [1]. Tenhamos em conta a implementação de um “256 tap filter” numa DSP. O processador de uma DSP convencional possui 1-4 MAC units, assim como “barrel shifters” e outros circuitos de otimização para cálculos eficientes [5].

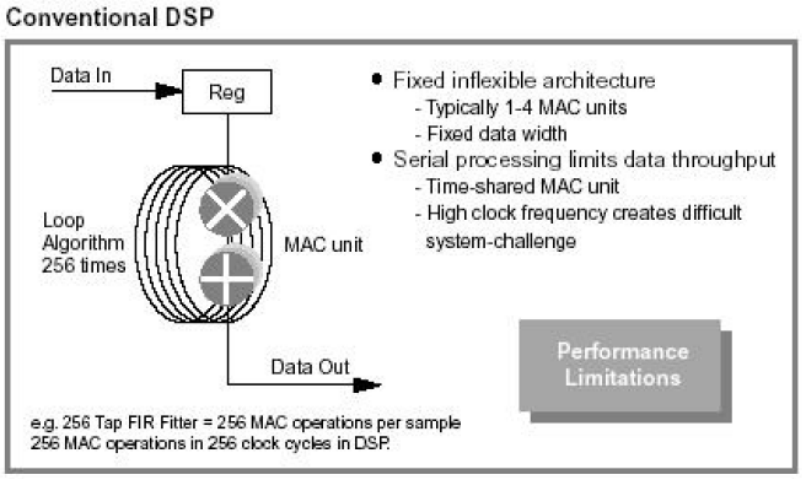


Figura 2.4 – Filtro DSP [5].

Assumindo que esta possui apenas 1 MAC unit, e que o "256 tap filter" envolve 256 MAC operations por amostra, seriam necessário 256 clock cycles para que o output fosse processado no processador da DSP. Para melhorar este sistema seria necessário aplicar um clock de alta frequência, aumentando a complexidade e o custo. Para além disso a probabilidade de ocorrer "clock skew" para frequências elevadas são também elevadas.

Analisemos agora o mesmo filtro implementado numa FPGA. Relembrando a característica de "paralelismo" e o seu número elevado de Gates e transístores, o mesmo filtro poderia ser implementado "em paralelo" como demonstra a figura seguinte, sendo necessários 256 registos e 256 multiplier units, juntamente com a soma final. Neste modo, a filtragem que demora 256 ciclos a ocorrer numa DSP, ocorre, numa FPGA, em apenas 1 ciclo. Como resultado, a latência de cada instrução é muito melhorada.

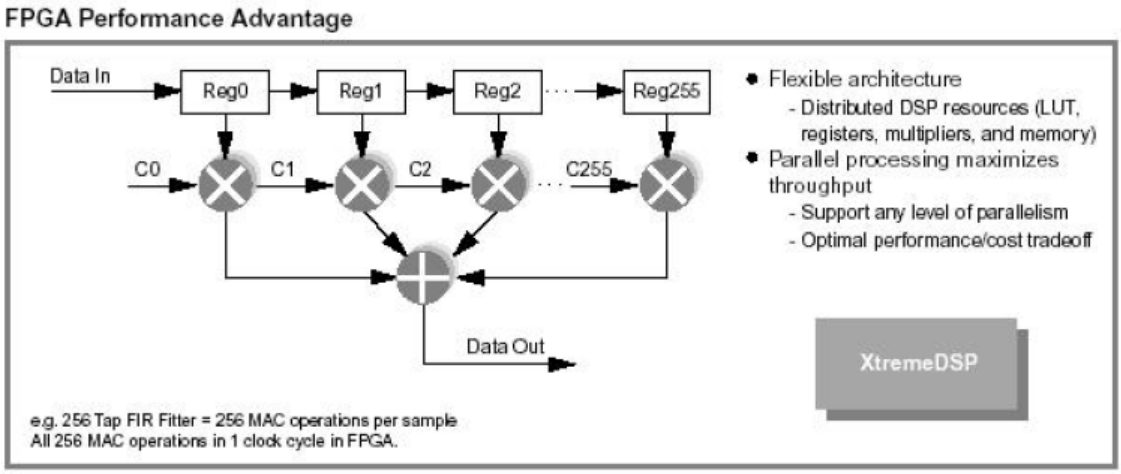


Figura 2.5 – FPGA – Flexibilidade na arquitetura + Processamento em paralelo [5].

A performance do cálculo das FPGAs é tão rápida que o "control loop rate" é apenas limitado pelos sensores, atuadores e módulos I/O. Para além disso, o valor do jitter depende da precisão do clock, rondando o valor de pico-segundos (processadores comuns: jitter: menos de 100milisegundos). A utilização da prototipagem em FPGAs, permite-nos utilzar outros tipos de programas, como o MATLAB, e usar controlos em VHDL ou Verilog para "fusão" das duas programações. Para além disso, uma típica FPGA consome menos potência que um controlador baseado em microprocessadores ou em ASIC [1].

2.3 Motores Elétricos

O sucesso do desempenho de um sistema robótico móvel está fortemente condicionado pelo motor elétrico e por toda a eletrónica de potência associada aos controladores e conversores de potência.

Tendo o motor elétrico como função principal a conversão de energia elétrica em energia mecânica (cinética), a aplicação de um conversor de potência é indispensável para fornecimento dos corretos valores de corrente e tensão do mesmo. Por sua vez, a lógica de controlo aplicada a este conversor pode ser realizada através de um controlador eletrónico. Este, através de leituras obtidas por sensores, realiza uma análise do estado do motor e corrige os valores de binário e velocidade do motor.

Uma estrutura robótica pode ser dividida em três unidades principais. A aquisição de sinais é tipicamente implementada por um conjunto de sistemas de sensores. Os sinais adquiridos pelos sistemas de sensores podem corresponder a uma grande variedade de parâmetros do motor (correntes, tensões, velocidades, posições, temperatura, etc.). Estes sinais passam por um processo de filtragem e adaptação, para que possam ser corretamente interpretados por um processador, tipicamente presente num micro controlador. Por sua vez, o controlador gera os sinais na sua saída, resultado do processo de análise implementado internamente. Os sinais de saída do controlador são então aplicados a um conversor de potência proporcionando assim o controlo do motor.

Na figura 2.6 está apresentada uma montagem típica de uma estrutura robótica com motores elétricos, podendo ser observado um esquema do que foi descrito no parágrafo anterior. Na figura 2.6 são também apresentados alguns exemplos de controladores, conversores de potência e motores elétricos.

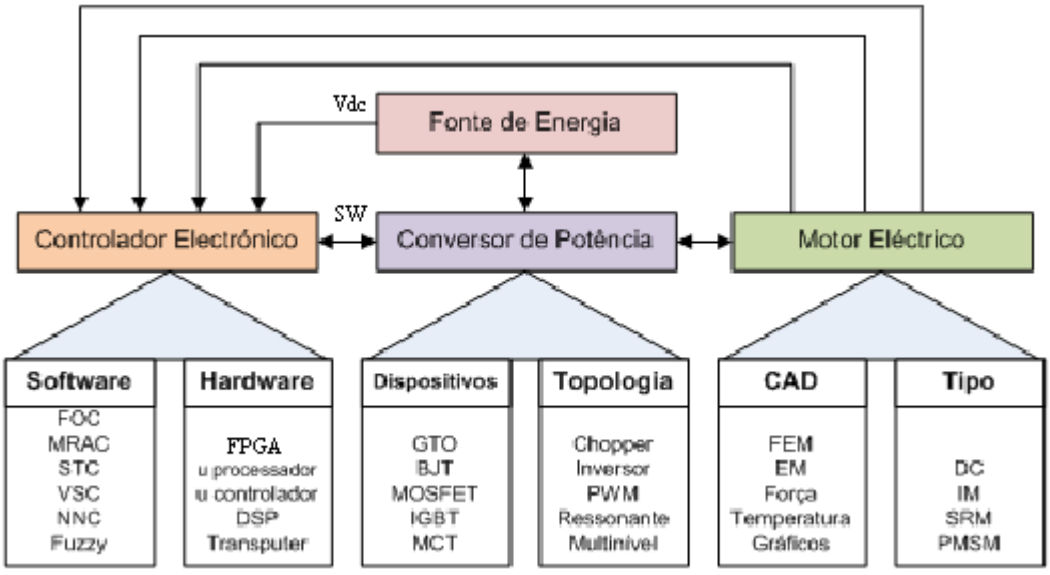


Figura 2.6 – Esquemático de uma montagem típica de uma estrutura robótica

São apresentadas de seguida alguns motores tipicamente utilizados em vários cenários da robótica móvel, sendo também realizada uma análise de alguns controlos típicos implementados em motores BLDC, procurando os requisitos mínimos da FPGA a aplicar no controlo do sistema.

2.3.1 Motor de Corrente contínua CC

Este tipo de motor é constituído por um enrolamento no rotor que roda livremente entre os polos do estator.

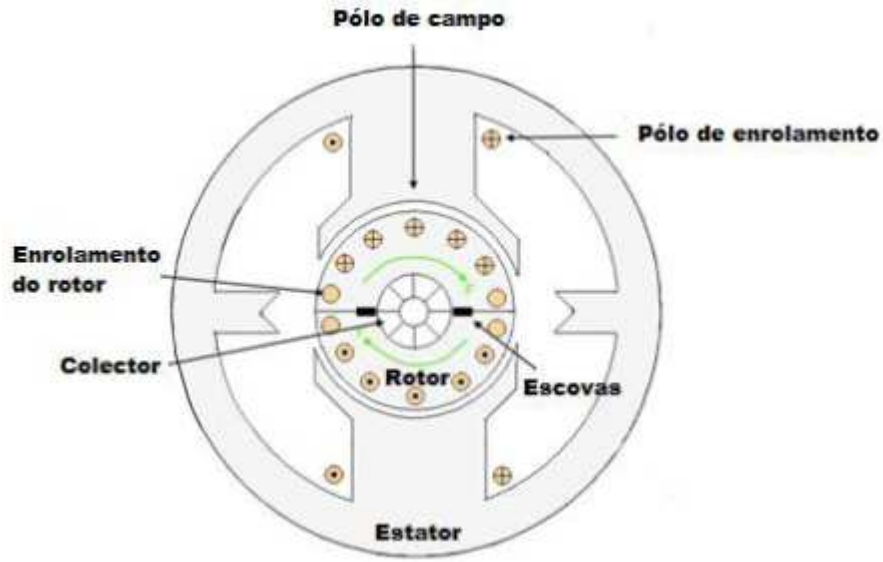


Figura 2.7 – Constituição do motor de corrente contínua [6].

Através da aplicação de uma corrente contínua ao enrolamento no rotor, por auxílio de escovas e coletor, o motor entra em rotação por interação entre o campo elétrico do rotor e o campo magnético do estator. A alternância do sentido da corrente aplicada ao rotor, provoca a rotação do mesmo, por intermédio dos contactos entre enrolamentos do rotor ao coletor [6].

No entanto, este tipo de motor é tipicamente volumoso, de baixo rendimento e fiabilidade e possui uma elevada necessidade de manutenção devido à presença de escovas, sendo por isso um motor mais utilizado para baixas potências [7].

Apesar da excelente regulação e controlabilidade do motor CC, este tem perdido muitos campos de aplicação nos acionamentos controlados em detrimento do motor assíncrono [8].

A constante evolução destes motores com inovações criativas e o auxílio de um processamento otimizado tem vindo a atribuir melhores características, tais como: melhor relação “torque/volume”, “torque elevado para uma vasta gama de velocidades”, “funcionamento suave para baixas velocidades” e “manutenção reduzida” [9].

2.3.2 Motor de Indução CA

O binário produzido pelo motor de indução depende da interação entre o campo magnético radial produzido pela corrente nos enrolamentos do estator, e pela corrente no rotor induzida por indução eletromagnética [10].

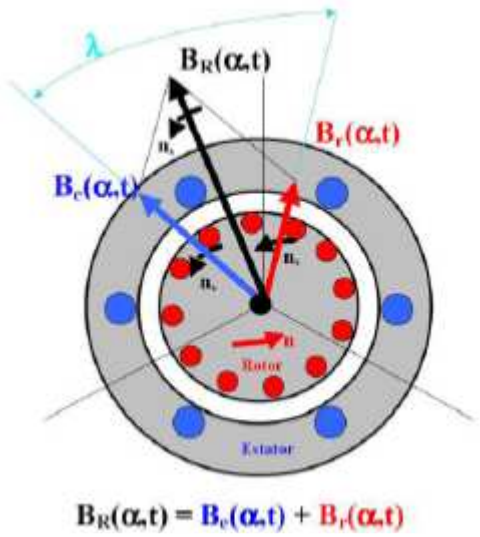


Figura 2.8 – Diagrama vetorial do motor de indução [10].

O enrolamento trifásico no estator encontra-se dentro de pequenas valas, podendo o rotor possuir um enrolamento trifásico ou um enrolamento em "gaiola de esquilo" [11].

A alimentação trifásica do estator provoca um campo magnético girante que induz uma força eletromotriz nos enrolamentos do rotor. O binário produzido por este motor é dependente desta mesma força eletromotriz. Neste tipo de motor, o rotor e o campo magnético girante do estator encontram-se a velocidades diferentes. Este fenómeno denomina-se "escorregamento" e é dependente da carga aplicada ao motor. Com o aumento de velocidade do motor, a diferença de velocidades entre o rotor e o campo girante do estator diminui, diminuindo assim a corrente induzida [11]. Estas velocidades são inversamente proporcionais ao número de pares de pólos do estator.

Os motores de indução são caracterizados pela sua simples construção, custo razoável, robustez, capacidade de operação em ambientes adversos e reduzida manutenção visto não possuírem escovas [11]. Apesar das vantagens mencionadas anteriormente, este motor apresenta elevadas perdas devido à utilização de enrolamentos no estator.

2.3.3 Motor Síncrono de Ímanes Permanentes

Os enrolamentos do estator no motor PMSM são semelhantes aos do motor de indução. Estes dois tipos de motores diferem na constituição do rotor, sendo o rotor do motor PMSM constituído por ímanes permanentes, que rodam ciclicamente provocando assim o campo magnético necessário ao funcionamento do motor. Este tipo de motores, não possuindo enrolamentos no rotor, são tipicamente mais leves, menos volumosos e de rendimento

superior (não existem perdas de Joule), quando comparados aos motores de indução. Devido à ausência de contato (sem escovas), o sistema de excitação deste tipo de motores não apresenta risco de dano mecânico, defeitos ou sobreaquecimento.

Apesar da sua estrutura simples, e das vantagens já mencionadas anteriormente (peso, volume, rendimento, manutenção), para além de terem um preço considerável, devido ao material magnético permanente, este tipo de motores têm associados a eles um tipo de controlo mais complexo.

2.4 Conversores de Potência

A evolução dos conversores de potência, tendem a seguir a evolução dos dispositivos de potência, procurando otimizar a sua eficiência, controlabilidade e confiabilidade.

Da grande variedade de conversores de potência existentes (AC-AC, retificadores, inversores, DC-DC), será dado especial ênfase aos conversores de DC-DC e DC-AC, com vista ao controlo de motores BLDC e PMSM.

Os conversores DC-DC surgiram na década de 60, constituídos por tirístores comutados a baixas frequências. Hoje, estes conversores evoluíram podendo ser agora operados por uma grande variedade de semicondutores e com altas frequências de comutação na ordem dos 100KHz.

Os conversores DC-DC permitem a conversão de energia de um nível de tensão inferior (ou superior), para outro nível de tensão superior (ou inferior), denominando-se assim "conversor DC-DC elevador" (ou abaixador). Como a potência fornecida ao conversor é igual à potência consumida, a variação dos níveis de corrente são inversamente proporcionais à variação dos níveis de tensão.

Em contraste com o conversor DC-DC, em que o valor da saída é mantido constante, no conversor DC-AC, o nível de tensão à saída corresponde a um valor oscilatório controlado (sinusoidal). A capacidade de alimentar uma carga em ambos os sentidos de corrente, concede a este tipo de conversores a denotação de "inversores".

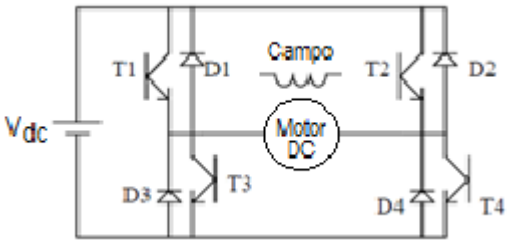


Figura 2.9 – Conversor de potência monofásico de ponte completa.

O conversor de potência presente na figura 2.9 denominado por "conversor monofásico de ponte completa", recorre à comutação de transístores aplicando diferentes níveis de tensão ao longo do controlo. Neste tipo de controlo, os mosfets de um mesmo braço têm sinais de controlo, opostos para que os terminais da alimentação não sejam curto-circuitados. Para garantir que dois mosfets nunca estão ativos ao mesmo tempo, é aplicado um "tempo morto". Durante este tempo ambos os mosfets de um mesmo braço têm um sinal de controlo igual a zero, garantindo-se a descarga ("abertura") do mosfet anteriormente fechado, podendo o outro mosfet fechar, sem perigo de ocorrer de curto-circuito da fonte de alimentação.

Apesar de ser possível alimentar estes tipos de inversores em tensão ou em corrente, devido à necessidade da aplicação de uma grande indutância em série ("fonte corrente"), este último tipo de alimentação tem vindo a cair em desuso. Para além disso, o grande valor de indutância influencia prejudicialmente na bidirecionalidade do fluxo de potência no inversor. A alimentação em tensão é então tipicamente uma preferência dos utilizadores.

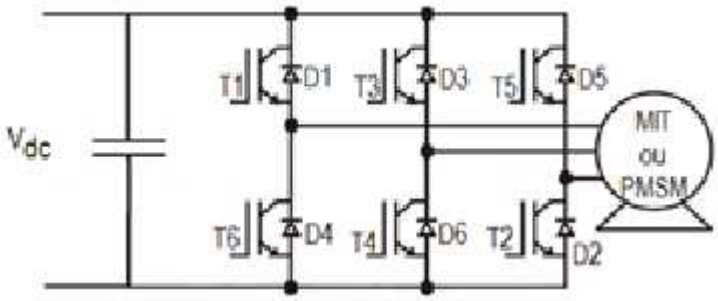


Figura 2.10 – Conversor de potência trifásico em ponte - inversor

Na figura 2.10 podemos observar um dos conversores de potência mais usados no controlo de motores trifásicos - ponte trifásica em H. Estes conversores são alimentados em tensão contínua e são capazes de fornecer níveis de tensão alternados à saída dos seus terminais. Há semelhança da ponte monofásica em H, este conversor recorre à comutação de transístores para realização do controlo do motor trifásico. A comutação e alternância dos valores de tensão à saída provocam a alteração dos níveis do campo magnético do motor.

No controlo típico de uma ponte trifásica em H para um motor de imãs permanentes, a comutação de um braço (troca de estados dos mosfets) ocorre sempre de 180º em 180º. No entanto, a existência de um terceiro braço proporciona comutações de 60º em 60º.

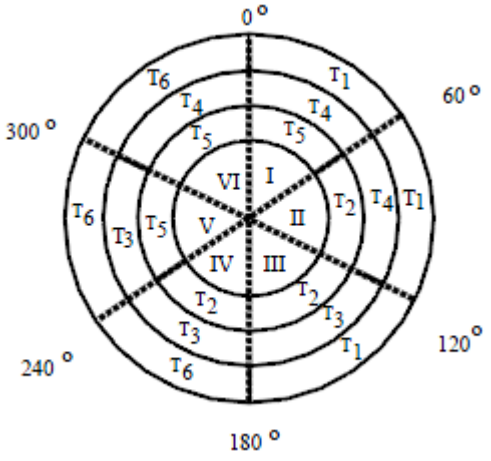


Figura 2.11 – Diagrama fasorial trifásico da ponte H e associação com respetivos transístores

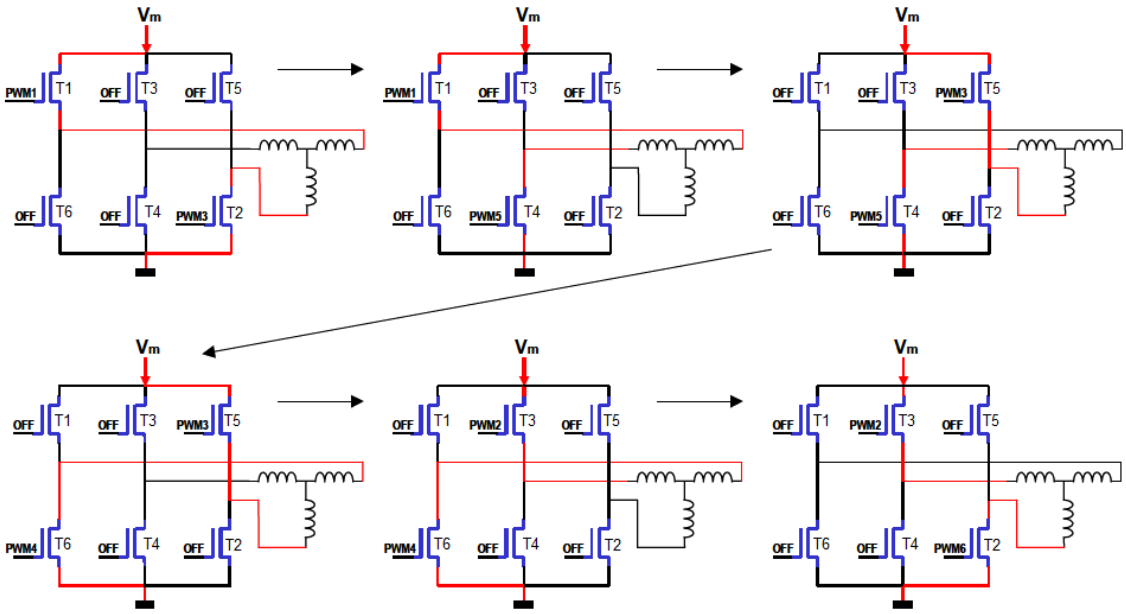


Figura 2.12 – Fluxo de corrente nos 6 estados de comutação – editado de [13]

Com auxílio da figura 2.11 e 2.12 podemos realizar uma análise dos estados dos mosfets numa ponte trifásica em H na realização do controlo de um motor BLDC.

Este conversor, ponte trifáscia em H, remete-nos para o próximo sub-capítulo, onde serão explicitadas algumas estratégias de controlo deste tipo de motores por intermédio de uma ponte trifásica em H.

2.5 Estratégias de Controlo

2.5.1 Space Vector Modulation

O método de controlo "Space Vector Modulation" tem vindo a ganhar cada vez mais popularidade devido à sua "universalidade", isto é, o controlo SVM pode ser implementado em motores de indução AC, motores brushless DC, motores de relutância e motores síncronos de ímanes permanentes.

Quando comparado com os métodos de controlo usuais, este permite uma tensão de saída de 15% superior (permitindo um uso mais eficiente da fonte de alimentação), maior eficiência e um menor número de comutações (cerca de 30%) quando comparado com o método convencional Sinusoidal Pulse Width Modulation [14].

Considerando um inversor trifásico em ponte, e admitindo que os três braços podem assumir dois valores diferentes ("1" se o transístor superior estiver saturado e o inferior em aberto; "0" se o transístor superior estiver em aberto e o inferior saturado), podemos obter 8 estados possíveis do conversor. Na tabela seguinte estão tabulados esses estados, e, para cada um deles, os valores de saída do conversor. Note-se que para os estados (1,1,1) e (0,0,0), os níveis de tensão de saída são iguais e nulos. Com estes oitos estados podemos realizar uma representação vetorial em estrela hexagonal (dois vetores nulos, representados no centro da estrela). Nesta estrela, qualquer vetor, pode ser obtido, pela soma dos dois vetores adjacentes.

Com a aplicação da transformada inversa de Park, conseguimos converter as componentes do vetor de tensão do referencial rotacional (Vd e Vq) num referencial estacionário do estator (V_α e V_β).

Tabela 2.1 – Os 8 estados de controlo de um inversor trifásico em ponte [15].

S_a	S_b	S_c	V_{AB}	V_{BC}	V_{CA}	SSV
0	0	0	0	0	0	V_{000}
1	0	0	V_{DC}	0	$-V_{DC}$	V_0
1	1	0	0	V_{DC}	$-V_{DC}$	V_{60}
0	1	0	$-V_{DC}$	V_{DC}	0	V_{120}
0	1	1	$-V_{DC}$	0	V_{DC}	V_{240}
0	0	1	0	$-V_{DC}$	V_{DC}	V_{300}
1	0	1	V_{DC}	$-V_{DC}$	0	V_{360}
1	1	1	0	0	0	V_{111}

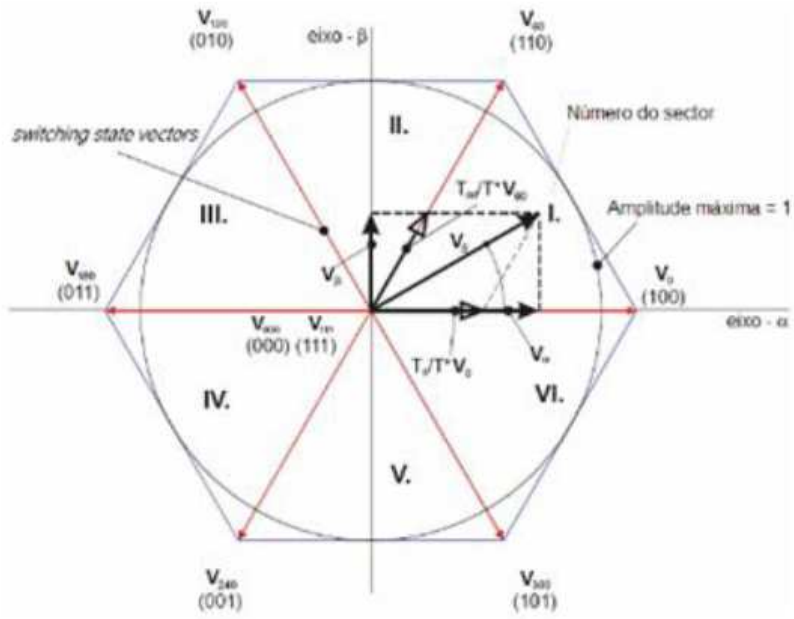


Figura 2.13 – Representação vetorial de um inversor trifásico em ponte – editado de [16]

2.5.2 Controlo por orientação de campo com base em sensores Hall

Outro dos métodos que tem vindo a tornar-se mais popular, em sistemas de menor precisão, é o controlo baseado por análise de sinais obtidos por sensores Hall.

Como se pode observar na figura, por utilização de sensores Hall, são estimados os valores de ω e θ_e (velocidade e posição do rotor). A velocidade estimada é subtraída à velocidade de referência, sendo o erro aplicado a um controlador PI, obtendo-se o valor de referência I_q. As correntes lidas I_a e I_b, e a corrente calculada I_c (soma das duas lidas), são convertidas para modelo ortogonal por intermédio da transformada de Park (bloco "dq/abc"). O erro das correntes ortogonais, I_q e I_d é então aplicado a dois PI's, originando os valores de V_d e V_q. Juntamente com o valor da posição do rotor θ_e, e as tensões V_d e V_q, é possível obter-se as formas de onda PWM, com uma lógica SVM, a aplicar ao inversor de tensão.

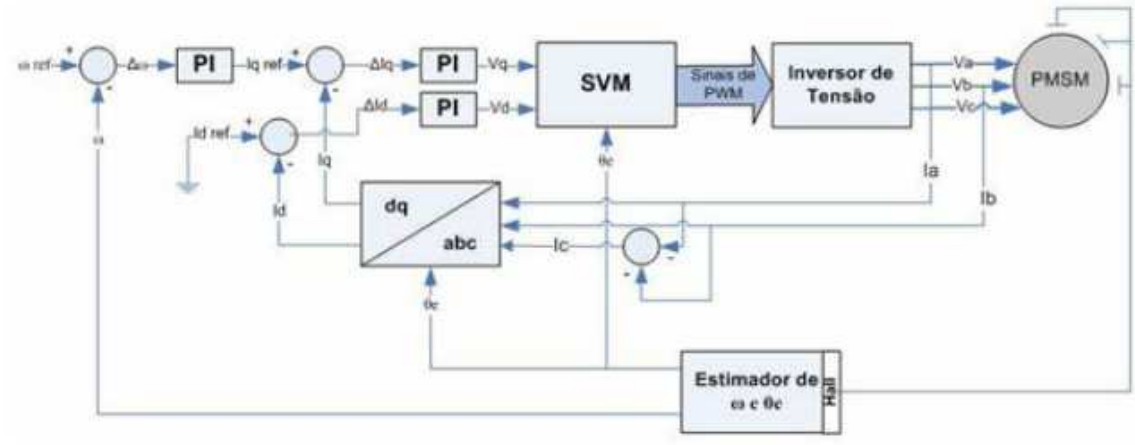


Figura 2.14 – Estrutura de controlo de um motor síncrono de ímanes permanentes [14].

2.6 Solução final

Ao contrário da maior parte dos controladores, cuja performance é influenciada pelo volume de código a implementar, a capacidade de processamento em paralelo de uma FPGA torna-a um interessante objeto de estudo para uma análise de performance, na medida em que a aplicação de um maior número de motores ou de uma lógica de controlo da estrutura robótica numa programação em paralelo, não irá influenciar substancialmente esta mesma performance. Por estes motivos, optou-se por um controlo da estrutura robótica a estudar implementado com auxílio de uma FPGA.

Devido ao seu elevado rendimento e baixo peso e volume, os tipos de motores escolhidos a aplicar na estrutura robótica são de imãs permanentes sem escovas, tipicamente denominados por BLDC (Brushless Direct Current Motor). Para além do preço elevado quando comparado com outros motores, a desvantagem do controlo ser mais complexo que o de outros motores apresenta-se nesta dissertação como um desafio a superar, testando a performance da FPGA.

O circuito de potência implementado é semelhante a uma ponte trifásica em H, constituído por "3 meias pontes". Os comandos dos mosfets serão aplicados diretamente de seis saídas da FPGA, sendo estipulados os valores de "tempo morto" e "DutyCycle" da pwm em código Verilog.

Para aquisição do valor de corrente consumida, é aplicada uma resistência R_{sense} entre a massa e as sources dos mosfets inferiores da ponte trifásica. A corrente consumida pelo motor (desprezando o valor de perdas por efeito Joule, residuais nos motor BLDC) circulará sobre a resistência R_{sense}, provocando uma queda de tensão proporcional à corrente. O valor da tensão, depois de adquirida pela FPGA, por intermédio de um ADC, será associada ao valor da corrente consumida pelo motor.

Quanto à lógica de controlo, este será realizado com auxílio dos sensores de posicionamento de efeito de Hall. Na busca de um binário máximo procurar-se-á aplicar uma fase de 90º entre os campos magnéticos do estator e do rotor. Poderá ser visto mais à frente, que irão ser realizadas comutações de 60º em 60º provocando uma oscilação deste desfasamento entre os 60º e os 120º, atingindo o máximo pretendido a meio do ciclo de comutação. O duty cycle da PWM a aplicar será calculado por intermédio de um primeiro controlador PI, para corrigir o erro de velocidade e por um segundo PI, para proteger o motor de altos valores de corrente (lidos em R_{sense}).

Foi escolhida uma estrutura robótica associado a um robot de 2 motores (2 rodas), mantendo no entanto alguma flexibilidade para a alteração da largura do robot, do raio das rodas e do número de motores (para 3 ou 4 motores/rodas). Com intuito do estudo de um método com maior precisão no processamento de um robot a seguir um percurso, foi escolhido uma metodologia de controlo capaz de privilegiar a correção do erro da velocidade linear ao erro da velocidade angular do robot.

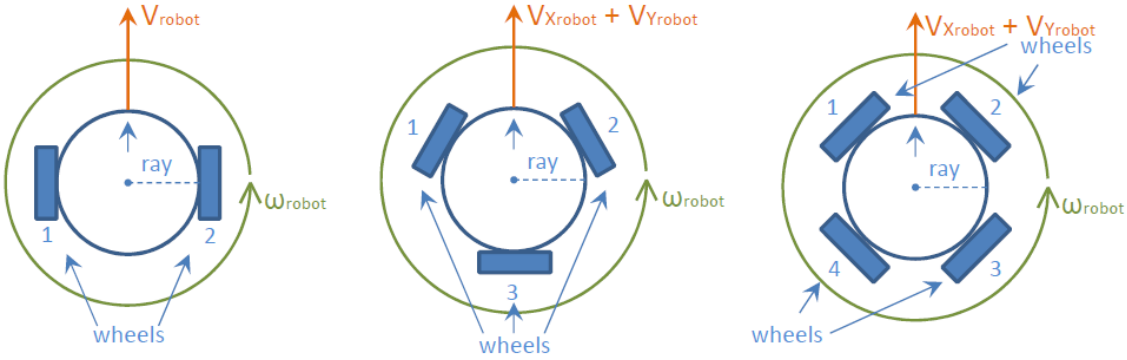


Figura 2.15 – Estruturas robóticas com 2, 3 e 4 motores

Capítulo 3

Programação da FPGA

Neste capítulo serão apresentadas as principais características da FPGA escolhida para implementação do controlo dos motores e do robot.

Será também exposto toda a lógica do programa implementado na FPGA em Verilog assim como o resultado de algumas simulações de funções elaboradas.

3.1 FPGA Spartan3E- GODIL

Após um levantamento de requisitos e possíveis aplicações futuras da estrutura robótica final notou-se que, apesar de algumas caraterísticas se manterem inalteradas (como o controlo da velocidade dos motores ou da velocidade linear e angular do robot), havia interesse em que outras características fossem flexíveis a alterações para adaptação em outras situações, como por exemplo:

- tipos de estruturas robóticas - o número de motores assim como a sua disposição influenciam o método de controlo do robot. Comparando duas estruturas robóticas com números de motores diferentes, rapidamente reparamos que as velocidades linear e angular do robot são obtidas por expressões diferentes;
- tipos de controlo do robot – A aplicação de diferentes tipos de controlo sobre o robot permite-nos estudar qual o melhor método de controlo. Como poderemos ver mais à frente, nesta dissertação foi implementado um método de control que privilegia a correção do erro da velocidade angular à correção do erro da velocidade linear. No entanto, também é possível controlar-se as velocidades dos motores independentemente das velocidades linear e angular do robot;

- tipos de motor – existindo uma grande variedade de motores aplicados na robótica móvel (BLDC, PMSM, servo motores…), deve também existir uma certa flexibilidade na escolha do motor presente na estrutura robótica;
- tipo de controlo dos motores - mesmo para cada motor, existe uma grande variedade de controlos (controlo por corrente constante, Space Vector Modulation …);
- tipo de comunicação – para alteração de valores de configuração e controlo (porta-série, USB, wireless, bluetooth);
- outras características mais expecíficas – métodos de posicionamento e odometria (sensores de efeito de hall, encoders, leitura de correntes); opção de aplicação de uma pwm Unipolar ou Bipolar assim como a configuração do valor do "tempo morto" a aplicar (tempo que ocorre entre a abertura de um transístor e o fecho do transístor oposto numa ponte H).

Assim, como poderemos observar ao longo deste capítulo, foi utilizada uma "programação em paralelo" e extruturada por "blocos" em Verilog com auxílio de uma FPGA concedendo-nos não só uma grande rapidez de tratamento de sinais, como a flexibilidade pretendida.

Na figura 3.1 podemos observar a estrutura interna do código final realizado.

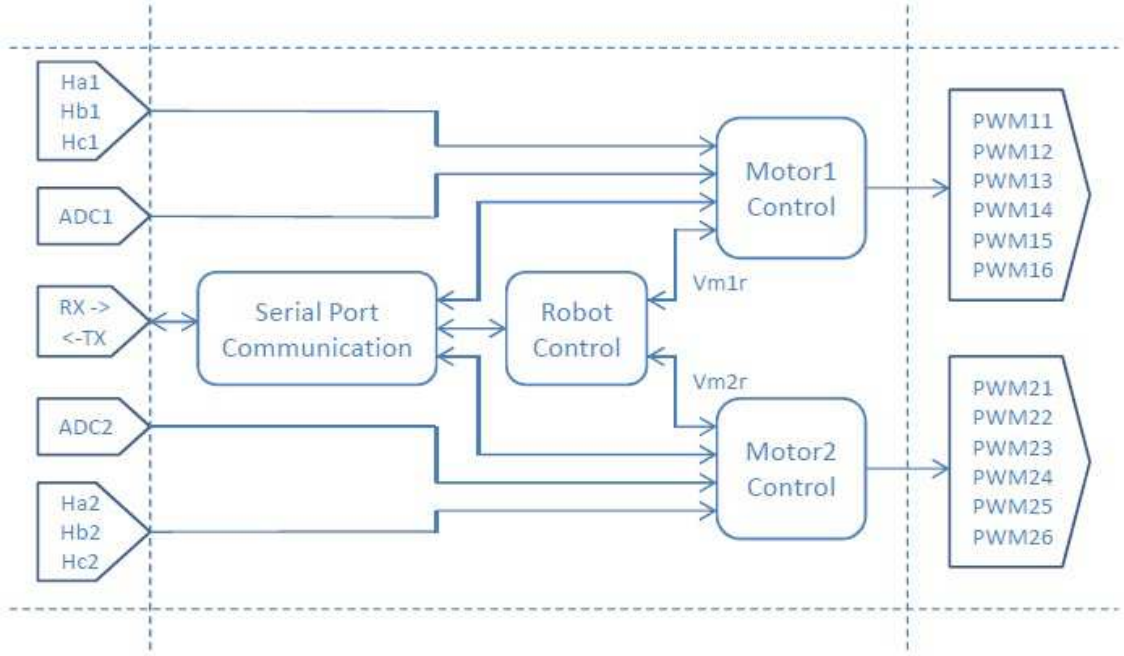


Figura 3.1 – Estrutura interna simplificada do código em Verilog

Podemos observar que a sua estrutura é constituída por quatro blocos principais internos associados ao método de controlo do robot, ao método de controlo dos motores e à comunicação realizada pela porta série. Do lado esquerdo estão apresentadas as entradas físicas do bit ADC (associado à leitura de corrente consumida, explicado pormenorizadamente neste capítulo), as entradas físicas dos sensores de efeito de hall dos motores BLDC e a entrada e saída para comunicação da porta-série (Rx, Tx). Do lado direito estão apresentadas todas as pwm a aplicar às pontes em H que alimentam os motores BLDC.

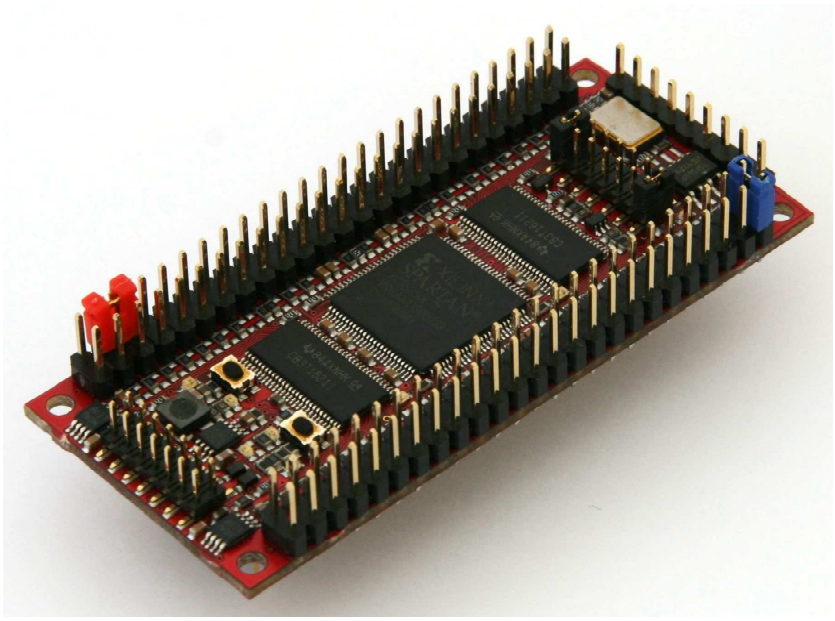


Figura 3.2 – Spartan3E – GODIL [17]

A FPGA utilizada foi uma Spartan-3E numa plataforma GODIL. Seguem-se algumas características desta plataforma que influenciaram na escolha para aplicação nesta dissertação [17] [19]:

- O seu grande número de pinos entrada/saída: 48 I/Os (+ 2 input clock) – tendo em conta que para o controlo de um motor são necessárias 4 entradas (3 sensores de hall, mais o bit ADC de corrente consumida) e 6 saídas (6 pwm a aplicar na ponte H que alimenta o motor), a aplicação desta FPGA numa estrutura robótica de 4 motores daria um total de 40 sinais. Somando mais 2 sinais respetivos ao Rx e Tx da porta-série temos um total de 42 numa FPGA de 48 I/Os;
- 20 dedicated advanced multipliers 18x18: importantes para a realização de cálculos matemáticos nas fases de controlo do robot e dos motores;
- Número de gates satisfatório: 500K;
- Alta frequência de clock: aproximadamente 50MHz (49,152MHz), permitindo distinguir eventos na ordem dos 20 ns - grande rapidez de resposta;
- Outros: Baixo custo, baixo consumo e dimensões reduzidas;

Com o conjunto de características apresentados anteriormente juntamente com o facto de procurarmos uma solução com o número de entradas e gates mínimo para um consumo também mínimo e, apesar de apresentar alguns resultados de performance inferiores quando comparadas com “Virtex2 pro” ou “Virtex4/5/6” [18], esta plataforma GODIL mostrou-se ideal para a aplicação desejada [17] [19].

3.2 Verilog – primeiros programas

Com intuito de familiarização com a plataforma da GODIL, foram realizados inicialmente alguns programas de complexidade variada que viriam mais tarde a ser utilizados na versão final do programa em Verilog.

3.2.1 Bloco controlador PI

Este bloco recebe como parâmetros de entrada, (para além dos sinais de clock da FPGA, enable e reset) os valores do ganho proporcional, ganho integral, "tempo de amostragem" deste controlador e um erro.

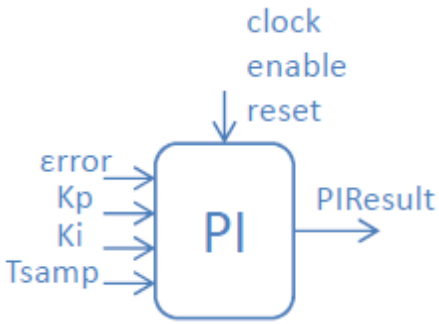


Figura 3.3 – Verilog – Bloco Controlador PI

O valor de saída deste controlador segue a expressão:

$$u(n) = K_p e(n) + K_i \sum_{k=0}^{n} e(k)$$

O valor à saída é limitado duas vezes, inicialmente quando ocorrer uma multiplicação (de um dosdois ganhos pelo erro) cujo resultado provoque overflow, e posteriormente realizando a soma dessas duas multiplicações. O valor máximo e mínimo de saída deste bloco correspondem aos valores máximo e mínimo com sinal de 32 bits.

Na figura 3.4 estão apresentados os resultados de uma simulação realizada com auxílio da ferramenta ISim, onde podemos observar o processo interno deste bloco.

São aplicados diferentes valores de erro ("error" - azul claro), para diferentes ganhos proporcional ("kp" – azul escuro) e integral ("ki" – azul escuro). A dourado conseguimos observar o resultado de PI ("piresult"), saturando nos dois limites. É possível também

visualizar algumas variáveis internas a este bloco, pintadas a cinzento, como o valor da multiplicação do erro pelo ganho proporcional (“presult”) e pelo ganho integral (“iresult”).

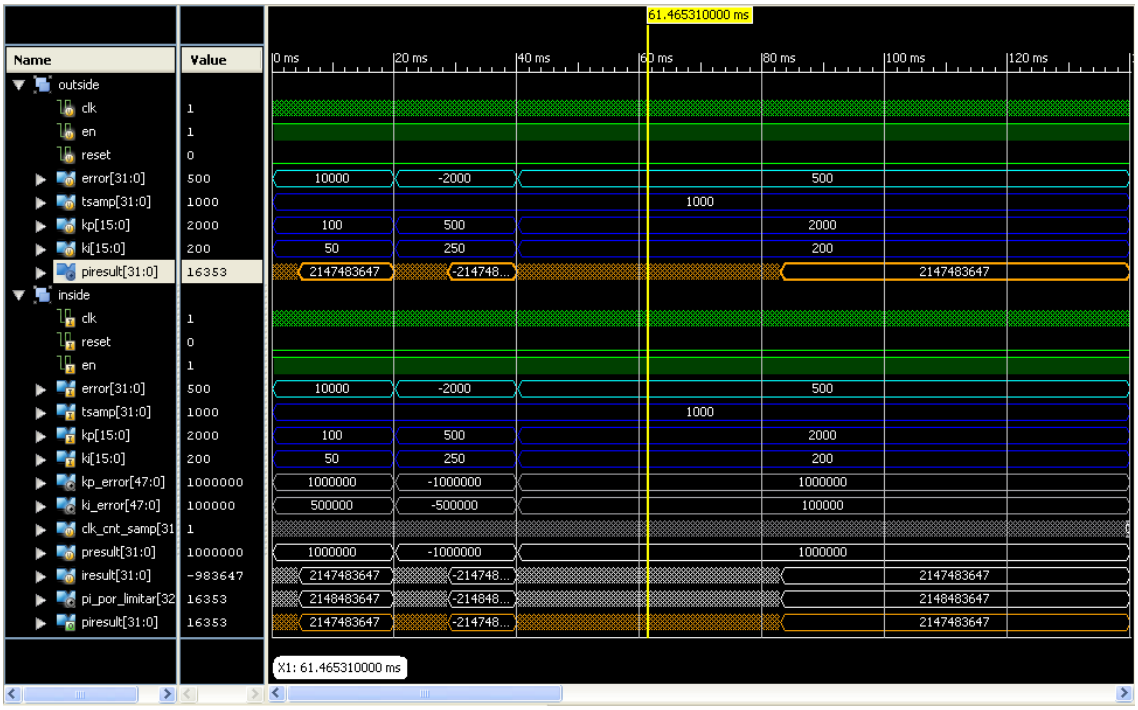


Figura 3.4 – Simulação – Bloco Verilog Controlador PI

Nos primeiros 20ms são aplicados os valores de 10000, 100 e 50 para o “error”, “K_p“ e “K_i“ respetivamente. O valor do tempo de amostragem (“T_{samp}”), é mantido constante ao longo da simulação em 1000 ns. Ao fim de aproximadamente 5ms conseguimos perceber que o valor de saída satura em 2147483647 (valor máximo, para um valor binário de 32 bits com sinal). Ao fim de 20ms o valor do erro é alterado para um valor negativo, dando início ao ajuste do novo valor do PI, saturando desta vez no seu valor mínimo. Na prática, espera-se que o valor de saída do bloco PI não atinja valores de saturação, no entanto, tais situações devem ser tidas em conta para que não ocorra overflow mantendo consistência ao valor de saída do PI.

Este bloco foi mais tarde utilizado em várias partes da versão final do código, como por exemplo, cálculo do valor do duty cycle a partir do erro da corrente.

3.3.2 Bloco gerador sinusoide

O primeiro método aplicado a este bloco foi um método por “previsão” do próximo ponto da sinusoide por intermédio das suas “derivadas discretas”.

O output (chamemos-lhe “seno”) é inicializado ao valor máximo pretendido e uma segunda variável interna (chamemos-lhe “cosseno”) inicializada a zero. A cada clock ao valor

do seno era subtraído o valor de cosseno ("escalado"/dividido por 2^3), e ao valor de cosseno o valor do seno ("escalado"/dividido por 2^3).

Para análise deste bloco foi associado em série um segundo bloco gerador de uma pwm. A aplicação da sinusoide na entrada deste segundo bloco provoca a oscilação do DutyCycle da onda quadrada. Na figura 3.5 podemos observar o resultado em osciloscópio da associação em série de um filtro passa-baixo, implementado eletronicamente.

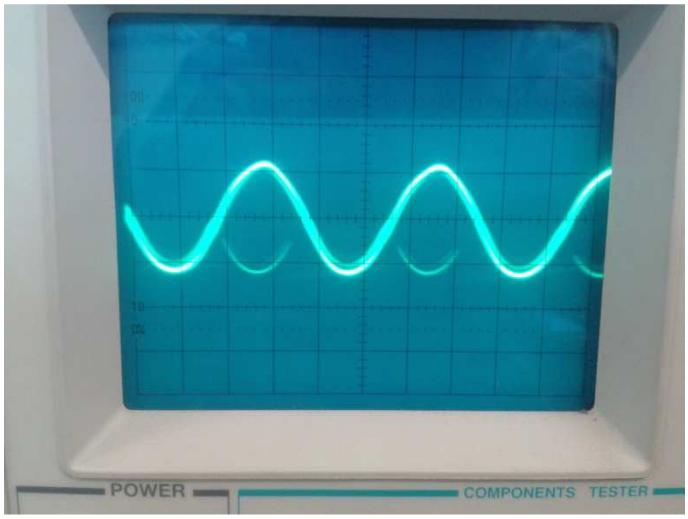


Figura 3.5 – Bloco sinusoide 1 (Osciloscópio)

Numa análise cuidada da figura 3.5 consegue-se reparar na estranha espessura e forma da onda sinusoidal. Esta "espessura" deve-se ao facto da onda sinusoidal não conter um "período fixo" (diferentes números de clock para diferentes ciclos sinusoidais), provocando pequenas oscilações da forma de onda e efeito de "relevo".

Como solução a este problema foi realizado um bloco capaz de adquirir valores guardados num ficheiro na pasta do projeto onde estavam guardados os valores da sinusoide (LUT). Aos valores de entrada correspondiam um fator multiplicativo e a frequência desejada da sinusoide.

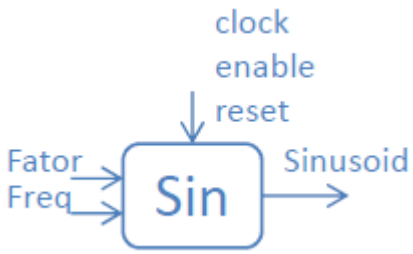


Figura 3.6 – Verilog – Bloco sinusoide

O período mínimo desta sinusoide, corresponde ao número de valores guardados no ficheiro multiplicado pelo período de clock da FPGA e a amplitude máxima corresponde ao máximo valor guardado no ficheiro multiplicado pelo parâmetro de entrada “fator”.

3.4 Bloco PWM

Este bloco possui como entradas os valores de “DutyC”(duty cycle), de Tdead (tempo morto) e Tswitch (período de comutação), e como saídas os valores de duas pwm (quase) simétricas. À saída são geradas duas ondas quadradas com período definido em “Tswitch” e Dutycycle (da saída “PwmHigh”) em “DutyC”.

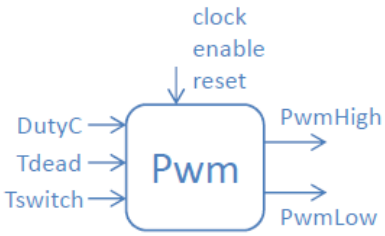


Figura 3.7 – Verilog – Bloco pwm

Relembrando que tencionamos realizar o controlo de uma ponte H de três braços, e que cada braço é constituído por dois mosfets de potência, foi realizada uma máquina de 4 estados que, não só diferencia os estados “High” e “Low” mas também um terceiro estado (que se repete nas transições de “High” para “Low” e vice-versa) em que ambas as pwm estão a zero garantindo assim que os mosfets nunca estão ambos fechados protegendo a fonte de curto-circuitos. Podemos identificar os seguintes quatro estados:

1. Num primeiro estado da máquina de estados (“PwmLow”=0; “PwmHigh”=1), um counter interno é incrementado à frequência do clock e comparado com o valor apresentado à entrada de “DutyC”. Quando o valor do counter iguala o valor de “DutyC”, é identificado um novo estado da máquina de estados;
2. Neste segundo estado, um segundo counter percorre uma temporização declarada em “Tdead” (“tempo morto” – tempo que ocorre entre a abertura de um mosfet e o fecho do seu oposto num mesmo braço de uma ponte H), mantendo ambas as saídas a zero. Quando o segundo counter iguala o valor de “Tdead”, o valor de “PwmLow” é colocado a 1, e o terceiro estado é identificado;
3. Neste estado, o valor do primeiro counter continua a incrementar até igualar o valor do período de comutação (definido em “Tswitch”). Quando estes valores igualam, o quarto estado da máquina de estados é identificado;
4. Neste estado ambas as saídas são colocadas a zero, e o segundo counter reiniciado, ocorrendo novamente o seu incremento até igualar o valor em Tdead.

Quando estes valores igualam, o valor de “PwmHigh” é colocado a 1 e o processo repete-se novamente desde o 1º estado;

Foram realizadas algumas simulações com auxílio da ferramenta ISIM, para confirmação do correto funcionamento deste bloco. Seguem-se algumas simulações para diferentes valores de entrada:

Figura 3.8 – ISim Simulador – Bloco pwm

Na figura 3.8 podemos observar a variação do duty cycle da pwm, para valores de 25%, 50%, 75% e 100% (linha azul na imagem: 12500, 25000, 37500, 50000, respetivamente, para um “TSwitch” de 50000). Os valores de “tempo morto” foram alterados ao mesmo tempo que os valores de duty cycle para 250, 500, 750 e 0 ns.

Foram também realizados testes com sucesso para períodos de comutação na ordem da décima e centésima do milissegundo (0,1 a 0,01 ms), 10kHz a 100kHz. No entanto, devido à frequência do cristal, com o aumento do valor da frequência, perde-se resolução na geração da pwm.

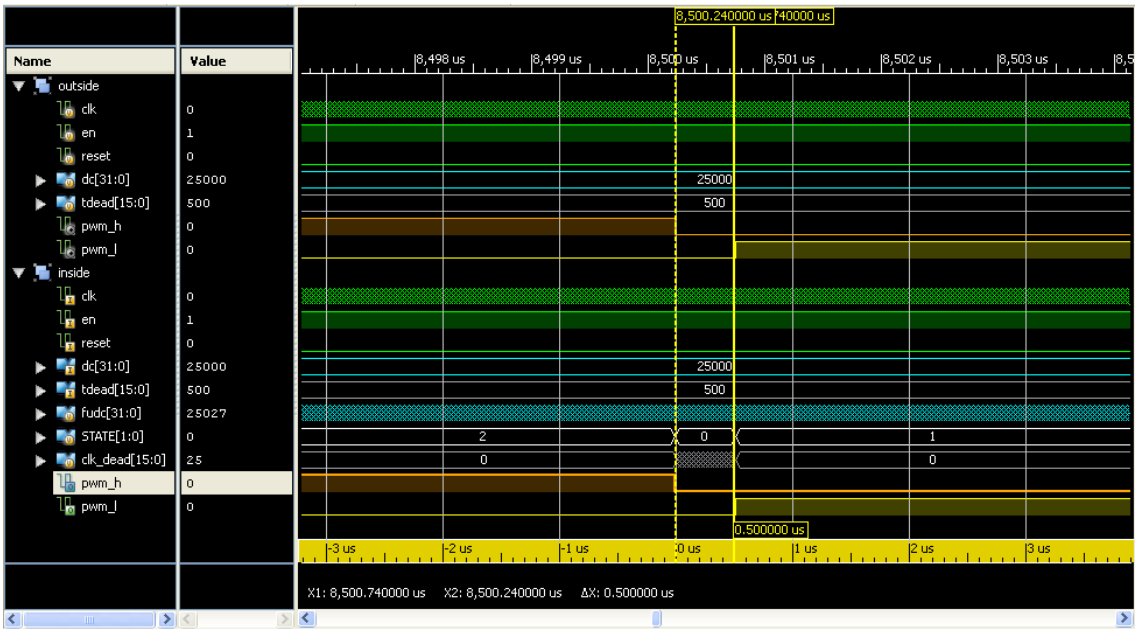


Figura 3.9 – ISim Simulador – Bloco pwm (análise do tempo morto)

Na figura 3.9 é apresentada a comutação das duas PWM. Foi escolhido o valor de 500ns para teste do valor de "tempo morto" (valor em que ambas as pwm estão a zero, para que se possa dar a descarga do mosfet anteriormente a 1, garantindo a descontinuidade de corrente no braço). Como podemos observar no fundo da tabela de gráficos, a diferença entre os dois cursores apresenta "0,500us", ou seja, 500ns.

Estes valores foram também comprovados eletronicamente por intermédio de um osciloscópio, como irá ser mencionado no capítulo de resultados.

3.3 FPGA –Código final Verilog

A estrutura final do código Verilog é constituída por 3 blocos principais, mais um bloco de comunicações, como pode ser visto na figura 3.1 e, mais pormenorizadamente, na figura 3.10.

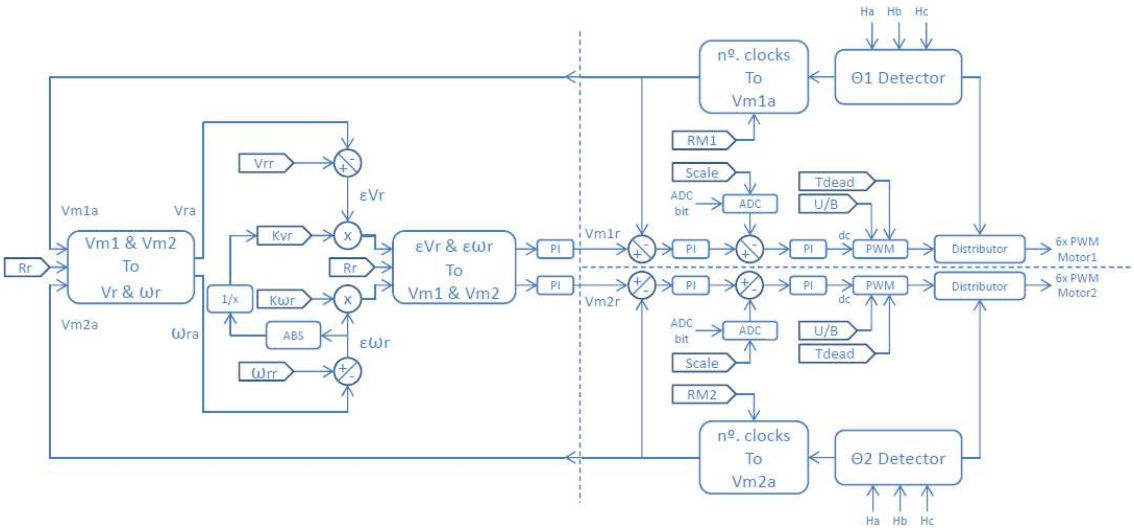


Figura 3.10 – Extrutura pormenorizada do código Verilog

São utilizadas algumas entradas e saídas físicas da FPGA associadas aos valores dos sensores de efeito de hall, um bit associado ao conversor ADC (que será explicado neste capítulo), os sinais de Rx e Tx da comunicação com a porta-série, e os 12 sinais de saída pwm (6 para cada motor).

3.3.1 Bloco Motor Control

Este bloco tem como função a implementação do controlo do motor BLDC. Como valores de entrada deste bloco temos o valor de referência da velocidade linear do motor (Vmr), os três sinais digitais dos sensores de efeito de hall, um bit pwm com duty cycle variável para o bloco de controlo de corrente, o raio da roda do motor, os valores de configuração dos controladores (Kp, Ki e Tsamp), entre outros parâmetros que serão mencionados mais à frente. Na saída temos o valor da corrente consumida e da velocidade atual do motor e 6 sinais digitais correspondentes aos sinais de comutação a aplicar ao driver de potência (ponte H).

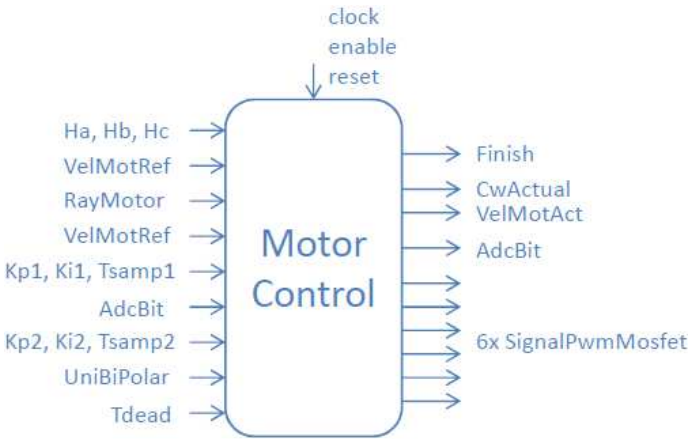


Figura 3.11 – Verilog – Bloco Motor Control

Este bloco é constituído por uma variedade de "sub-blocos":

Θ Detector – recebe como parâmetros de entrada os sinais digitais dos 3 sensores de efeito de hall e realiza uma contagem entre comutações do estado dos sensores; retorna também o sentido de rotação do motor;

Conversor de "clock counter" para velocidade atual do motor - converte o nº de clocks ocorridos entre comutação de sensores de hall / 60º em velocidade atual do motor;

PI – controlador Proporcional + Integrador

ADC – que recebe como valores de entrada uma pwm com DutyCycle proporcional ao valor de corrente consumida pelo motor;

PWM – gera um sinal pwm com período e duty cycle definido;

SixStepCommutator – que gera 6 sinais pwm a distribui aos 6 mosfets na ponte H;

3.3.1.1 Θ Detector

Para o controlo de malha fechada da velocidade do motor é necessário a leitura da velocidade do mesmo. Esta leitura pode ser realizada de várias maneiras, com auxílio de uma variedade de sensores ou leituras (encoders, sensores de efeito de hall, leitura da força contra eletromotriz, etc). Tendo em conta que o motor utilizado possui sensores de efeito de hall, foi realizado um programa capaz de calcular a velocidade do motor por intermédio destes mesmos meios.

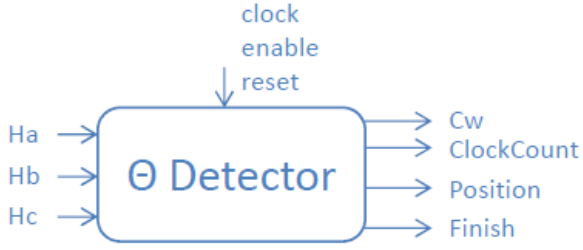


Figura 3.12 – Verilog – Bloco Θ Detector

Este bloco contém um counter interno que incrementa até detetar uma comutação nos sensores de efeito de Hall. Quando tal acontece, o valor do counter interno é reinicializado e os valores de saída "ClockCount", "Cw" e "Position" são atualizados, sendo também o bit "Finish" colocado a um. Este bit vai desencadear o início do cálculo da velocidade atual no bloco seguido a este.

A frequência do clock na entrada e o número de bits do counter interno deste bloco influenciam a precisão da frequência de rotação do motor e, por consequência, o valor da velocidade lida. Por exemplo, para um counter de 8 bits e uma frequência de ≈50MHz (frequência do cristal da FPGA), ou seja, um período de ≈20ns, o período mínimo e máximo entre comutações de 30º seria de 20ns e 5100ns (5,1 µs) entre comutações. Como poderemos observar na descrição do bloco seguinte, a estes intervalos correspondem as velocidades máxima e mínima de 5×10^8rpm e $1{,}96\times10^8$rpm, respetivamente. Como podemos observar, estes valores não cobrem as velocidades típicas usadas em robótica móvel. Torna-se então necessário escolher uma resolução do counter que nos permita, não só abranger valores de velocidade mais baixos como distinguir dois valores de velocidade relativamente próximos, garantindo uma precisão satisfatória. Optando por um counter de 32 bits conseguimos diferenciar velocidades na ordem dos 0,12rpm, ou seja, aproximadamente 8 minutos e meio por rotação.

3.3.1.2 Conv ClockCount VelMotActual

Este bloco realiza a conversão do número de clocks ocorridos entre comutação de sensores de efeito de hall, para a velocidade linar do motor.

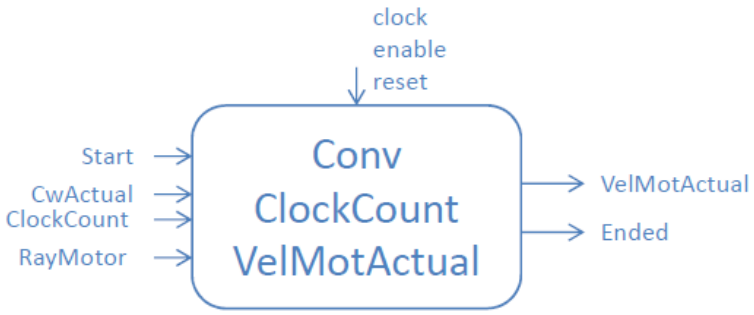


Figura 3.13 – Verilog – Bloco Conv ClockCount VelMotActual

O cálculo interno realizado é o seguinte:

$$V_{M1rpm} = \frac{60 \times 60}{N_{clk} \times 360°} \times f_{clk}$$

$$= \frac{10}{20ns} \times \frac{1}{N_{clk}} = \frac{5 \times 10^8}{N_{clk}}$$

Em metros por segundo (m/s) temos (com o valor do raio da roda definido em metros):

$$V_{M1linear} = \left(\frac{60 \times 60}{N_{clk} \times 360°} \times f_{clk}\right) \times (2 \times \pi \times r_{roda})$$

O conjunto de expressões anteriormente apresentadas são válidas para um motor brushless de 1 par de polos, ou seja, à velocidade do motor corresponde a velocidade do campo magnético girante. Para motores com mais de 1 par de polos, as seguintes expressões devem ser consideradas:

$$rotações\ do\ veio = \frac{rotações\ do\ campo\ magnético}{nº\ par\ de\ polos}$$

Assumindo "p" como o número de par de polos temos:

$$V_{M1rpm} = \frac{60 \times 60}{N_{clk} \times 360°} \times f_{clk} \times p$$

$$V_{M1linear} = \left(\frac{60 \times 60}{N_{clk} \times 360°} \times f_{clk} \times p\right) \times (2 \times \pi \times r_{roda})$$

3.3.1.3 Division

Uma das dificuldades deparadas na programação em Verilog, foi a realização de operações de dividir. Apesar de existir o operador de divisão ("/") em linguagem Verilog, este não é sintetizado em software da Xilinx, devido à grande dificuldade na conversão deste operador em componentes físicos da FPGA (gates, multiplicadores, flipflops, etc).

Na conversão das velocidades dos motores para velocidades linear e angular do robot (e vice-versa) é necessário essa mesma operação.

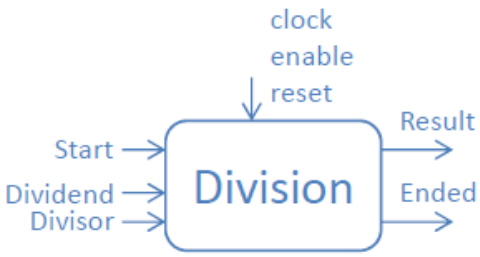


Figura 3.14 – Verilog – Bloco Dividir

O bloco “dividir”, recebe como entradas os valores do dividendo e divisor, começando o processo da divisão quando receber um impulso em “start”.

Inicialmente os valores são adquiridos para valores internos ao bloco, identificando se o divisor, ou dividendo são iguais a zero. No caso de o divisor for igual a zero, o valor do resultado é saturado a um valor limite. No caso de o dividendo ser igual a zero, o resultado é igualado a zero. Nestas dois casos, não é realizado nenhum cálculo, por isso o resultado à saída surge, no segundo clock.

Para o caso em que os valores lidos na entrada forem ambos diferentes de zero, é iniciado um processo em que são adquiridos os valores absolutos das entradas, assim como os seus respetivos sinais.

Em cada ciclo de clock, é realizado um “shift”, do bit mais significativo do dividendo para o bit menos significativo de uma variável “resto” (de igual número de bits). Com o novo valor na variável “resto”, esta é comparada com o valor em “divisor”. Se o valor de “resto” for menor que o valor em “divisor”, no clock seguinte, o novo valor do bit menos significativo em “dividendo” será zero. Se o valor de “resto for maior que o valor em “divisor”, então, no clock seguinte, ao valor do resto será subtraído o valor de “divisor”, e o novo valor do bit menos significativo de “dividendo” será um. Quando ocorrer um número de clocks igual ao número de bits do “dividendo”, esta variável conterá não o valor do dividendo adquirido, mas o resultado da divisão, o quociente. Depois de realizada a divisão, o valor é apresentado à saída, realizando-se os ajustes de sinal. Juntamente com o valor na saída é enviado um impulso no bit “ended”, indicando que o valor à sua saída foi atualizado.

Na imagem seguinte podemos observar um exemplo de uma divisão binária:

N: 13744/19=723(+resto=7)

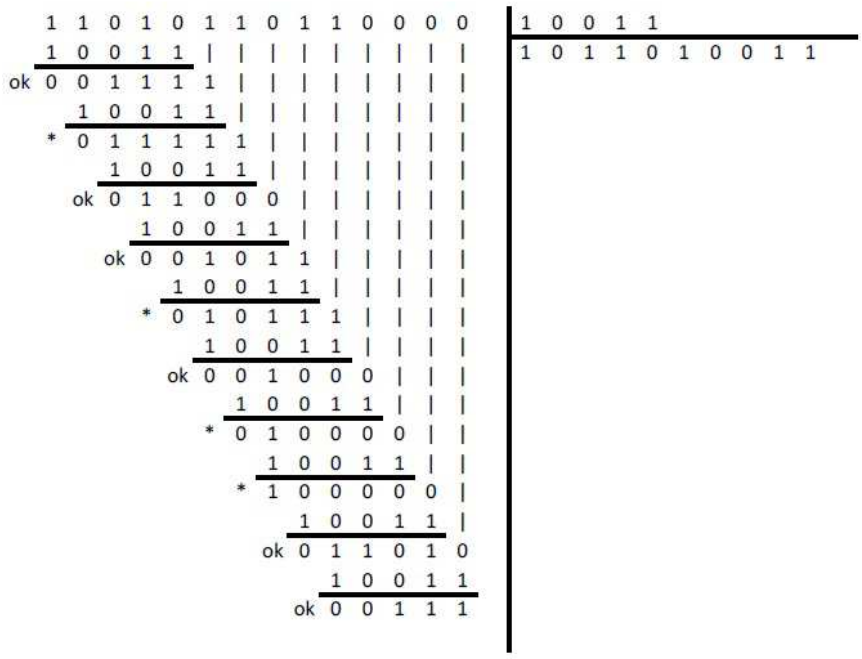


Figura 3.15 – Bloco Dividir – Divisão Binária

Foram também realizadas simulações ISIM para este bloco, não só para confirmação dos resultados esperados como para análise do tempo de resposta.

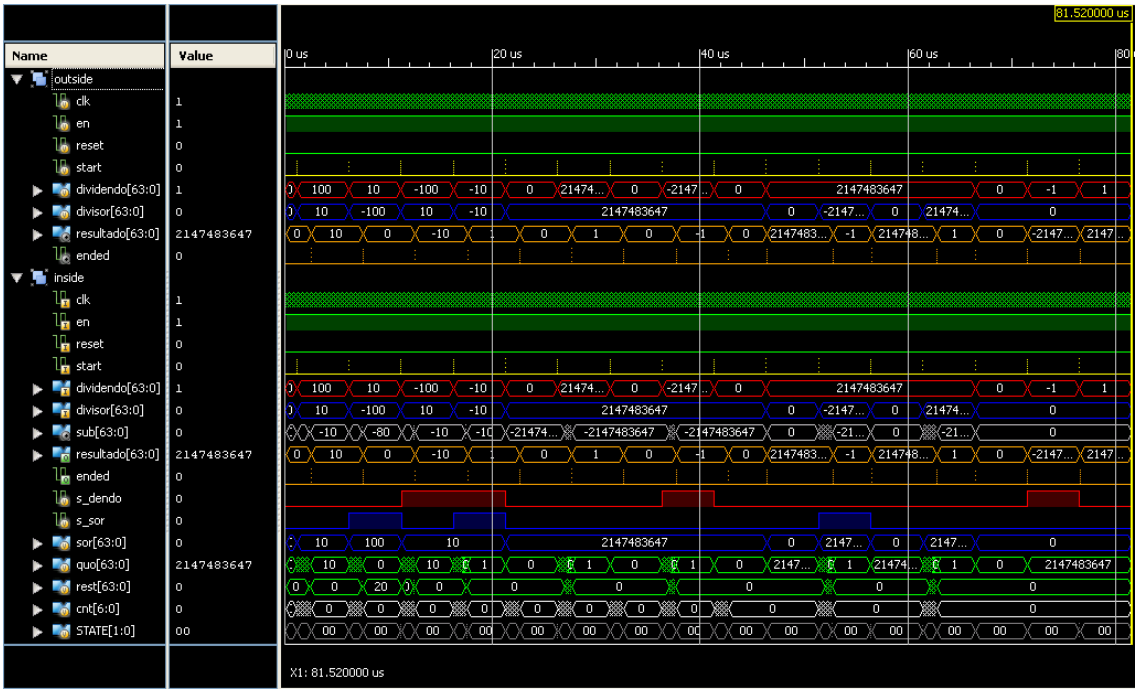


Figura 3.16 – Bloco Dividir – Simulação1

Na figura 3.16 podemos observar a atualização do resultado na saída, para todas as situações mencionadas anteriormente (divisão por zero, etc.).

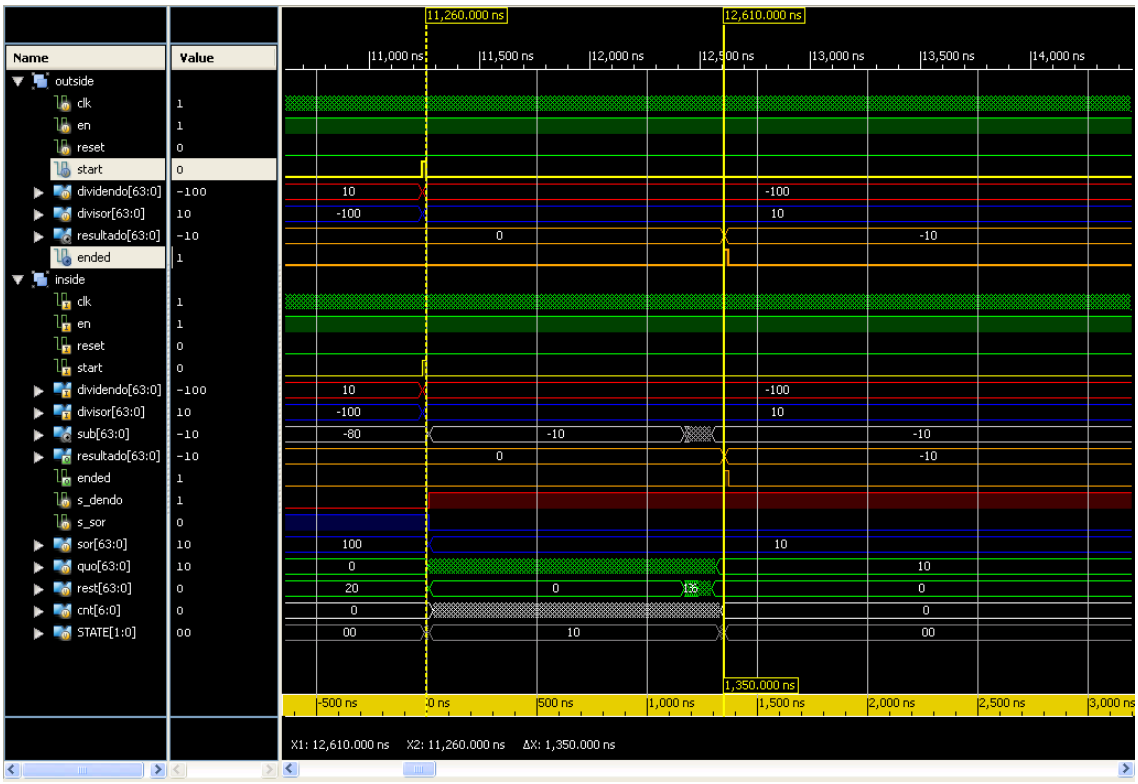


Figura 3.17 – Bloco Dividir – Simulação2

Como poderemos ver mais à frente, ao contrário das dificuldades deparadas com o processo da divisão, o processo de multiplicação está apenas limitado ao número de multiplicadores da FPGA.

3.3.1.4 PI

O bloco PI é usado como controlador do erro de velocidade e de corrente. O erro da velocidade, resultado da subtração do valor da velocidade de referência com a valor da velocidade atual (calculado pelo bloco "Θ Detector") é aplicada a um PI, obtendo-se uma referência de corrente. Esta nova referência, é subtraída com a leitura recebida pelo ADC, e aplicada novamente num novo PI, obtendo-se o valor DutyCycle a aplicar no bloco de PWM.

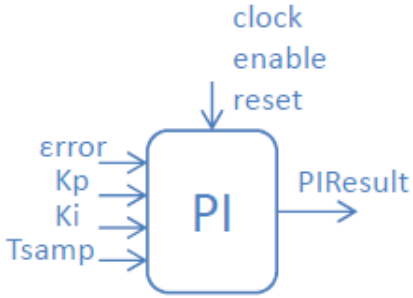


Figura 3. 18 – Verilog – Bloco PI

Não será realizada uma análise do funcionamento interno deste bloco, visto esta já ter sido feita num capítulo anterior.

3.3.1.5 ADC

Este bloco retorna o valor da corrente consumida pelo motor. A entrada "AdcBit" corresponde na verdade a um pino físico da FPGA declarada como entrada. A saída retorna o número de clocks que o valor de "AdcBit" esteve a "1" que, depois de multiplicado por um valor de escala, resultará no valor de corrente consumida.

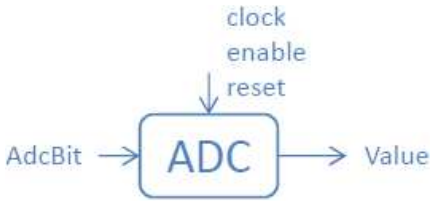


Figura 3.19 – Verilog – Bloco ADC

O circuito presente na figura 3.20 produz uma pwm de dutycycle proporcional ao valor de tensão lido na ponta de prova vermelha. Ao aplicarmos o valor de tensão lido na resistência R_{sense} da ponte em H, podemos converter um nível de tensão em "duty cycle" de onda quadrada. Aplicando a saída deste circuito à FPGA é possível obter o valor de tensão em R_{sense}, por intermédio do bloco "ADC".

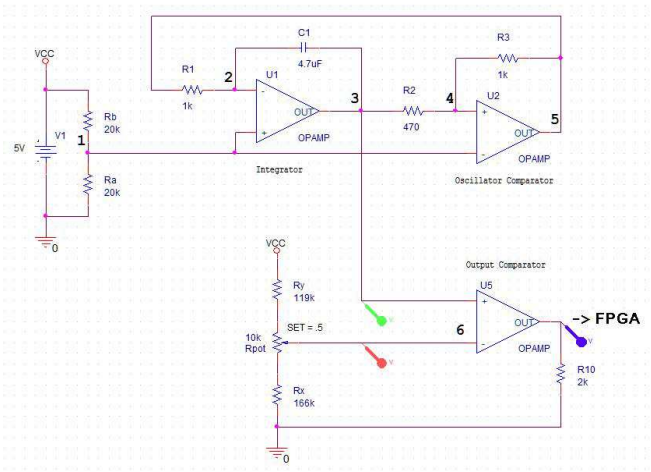


Figura 3.20 – Esquemático inicial Ampop PWM generator

Por análise da figura anterior, apercebemo-nos que a tensão V_3 no condensador C_1 resulta da seguinte expressão:

$$\frac{V_5 - 2.5}{R_1} = I_{C1} = C_1 \times \left(\frac{d(2.5 - V_3)}{dt}\right)$$

$$V_3 = \left(\frac{2.5 - V_5}{R_1 \times C_1}\right) \times t + 2.5$$

O segundo ampop está montado como um comparador, saturando a saída a dois níveis de tensão – 0 e 5V. Assumindo que a saída deste ampop é constante nessas duas situações, podemos afirmar que o valor de V_3 aumenta ou diminui consoante o valor de V_5 seja superior ou inferior a 2.5. Estas duas retas resultam numa forma de onda triangular com constante de tempo $\sigma=R_1\times C_1$.

Podemos também identificar os valores de pico da onda triangular da seguinte forma:

$$\frac{V_3 - 2.5}{R_2} = \frac{2.5 - V_5}{R_3}$$

$$V_3 = \left(\frac{R_2}{R_3}\right) \times (2.5 - V_5) + 2.5$$

Os valores de tensão de transição em V_3 podem ser obtidos pela substituição de V_5 pelos níveis de tensão ao qual satura (0 e 5V).

Por fim, aplicando um terceiro ampop como comparador de V_3 e a tensão aplicada em R_{sense} V_{sense} (na figura V_6), obtemos um sinal PWM à sua saída, com duty cycle variável e proporcional à tensão aplicada.

No entanto, na ideia de otimização do circuito e redução de custos, foi realizada uma pequena alteração do circuito implementando uma lógica um pouco diferente.

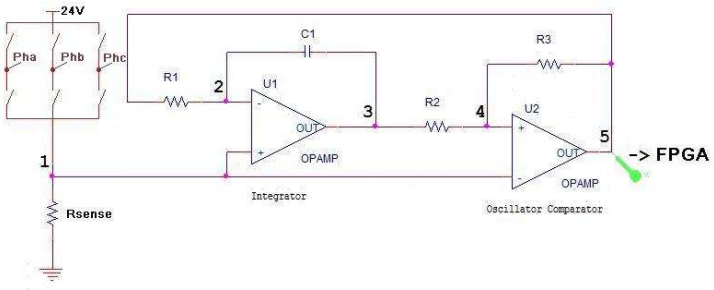


Figura 3.21 – Esquemático alterado Ampop PWM generator

Aplicando a tensão de R_{sense} na entrada do ampop integrador, provocará uma alteração na forma de onda na saída do "ampop oscilador comparador".

Nas imagens seguintes podemos observar que existe uma onda quadrada de dutycycle variável para uma banda de valores lidos aos terminais da resistência Rsense. Fora desta banda, a onda quadrada passa para o estado alto quando a tensão de Rsense for abaixo de um valor mínimo, passando a 0 quando superior a um valor máximo de tensão em Rsense.

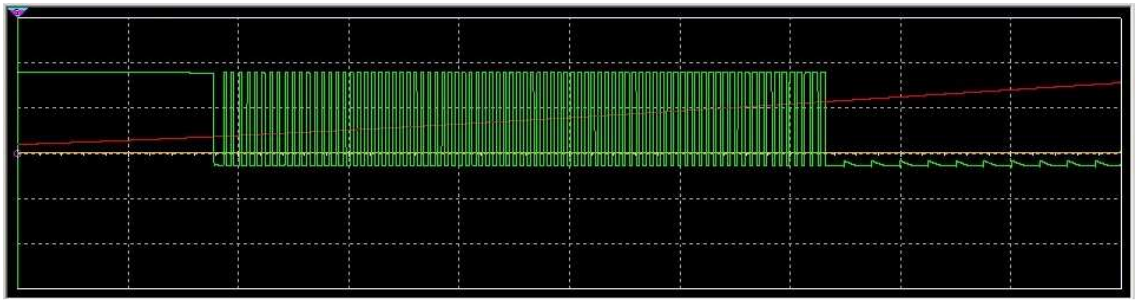


Figura 3.22 – PWM circuito – sinal PWM (a verde), sinal de controlo de DutyCycle (a vermelho)

Esta nova análise permite-nos distinguir 3 fases de consumo de corrente do motor: quando a corrente for mínima (não prejudique circuito eletrónico ou o motor) o valor da corrente não é tido em conta para cálculo do pwm aplicado à ponte trifásica em H; numa segunda etapa, o valor da corrente encontra-se entre dois valores (mínimo e máximo), influenciando no valor de dutycycle da pwm da ponte trifásica em H; por fim, quando a corrente ultrapassa um valor máximo, o valor de saída satura, minimizando o valor de DutyCycle da pwm da ponte trifásica em H; outro aspecto interessante neste novo controlo é a lógica inversa, pois ao contrário do circuito anterior, em que o valor da corrente era lido quando a onda quadrada de saída se encontrava a "1", neste circuito a leitura é realizada quando se encontra a "0", isto torna-se útil para situações de avaria – na falha de sinal ou de alimentação deste ampop, o valor lógico "0" corresponde a uma leitura de corrente máxima, minimizando o PWM da ponte trifásica em H, cortando a alimentação ao motor.

3.3.1.6 PWM

Este bloco produz uma onda quadrada com frequência definida em "Tswitch" e duty cycle em "DutyC" definidos como entradas. O valor de T_{dead} altera o valor do "tempo morto" entre as comutações das duas PWM que, é usado no andar de potência para garantir que um mosfet de um braço da ponte H não feche antes do seu complementar estar garantidamente aberto.

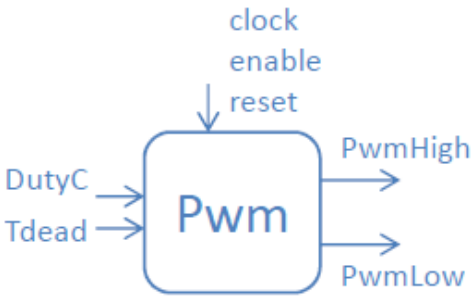


Figura 3.23 – Verilog – Bloco pwm

Não será realizada uma análise pormenorizada do funcionamento interno deste bloco, visto esta já ter sido feita num subcapítulo anterior.

3.3.1.7 Distributor

Este bloco tem como função a distribuição dos comandos pwm pelas 6 gates de cada mosfet.

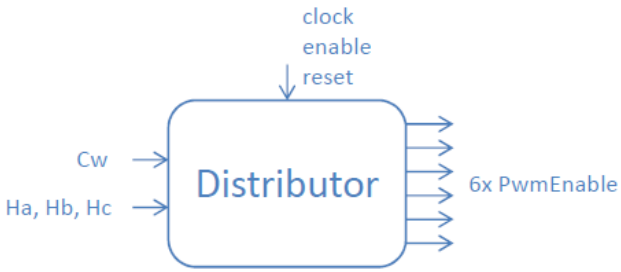


Figura 3.24 – Verilog – Bloco Distributor

Identificando os estados dos sensores de efeito de Hall, juntamente com o sentido de rotação do motor pretendido, este bloco replica e distribui a pwm pelas suas 6 saídas, provocando a comutação dos mosfets na ponte trifásica em H. A correta distribuição das pwm, permite ao motor funcionar num regime de máximo binário.

Na imagem seguinte podemos observar o valor do binário oscilar, atingindo o seu valor máximo aos 90º.

Figura 3.25 – Ripple do Torque de um BLDC

3.3.1.8 Odometry

Outra das funcionalidades que se demonstrou interessante analisar seria a odometria de cada motor. Com intuito de satisfazer esta necessidade foi realizado o bloco apresentado na figura 3.26, “Odometry”.

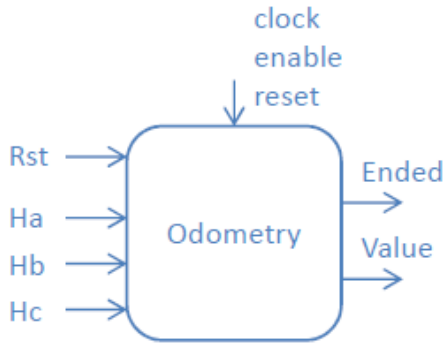


Figura 3.26 – Verilog – Bloco Odometry

Este bloco recebe como entradas os estados dos sensores de efeito de hall. Internamente, a combinação dos estados dos sensores é identificada e, na próxima comutação destes, o valor na saída é incrementado ou decrementado, consoante a rotação do rotor tenha sido no sentido dos ponteiros do relógio ou contra. A entrada "Rst" (Reset), é aplicada a um sempre que ocorre uma transmissão pela porta de série, provocando um reset no valor de odometria interno deste bloco. O valor de odometria é alterado no buffer de envio da FPGA para a aplicação gráfica, sempre que ocorre uma alteração do seu valor. Quando ocorre uma comunicação, é enviado o valor mais recente de odometria.

3.3.1.9 Motor Control – Arquitetura interna

Depois de todos os blocos serem testados, a estrutura interna pormenorizada do bloco "Motor Control" é a apresentada na figura 3.27.

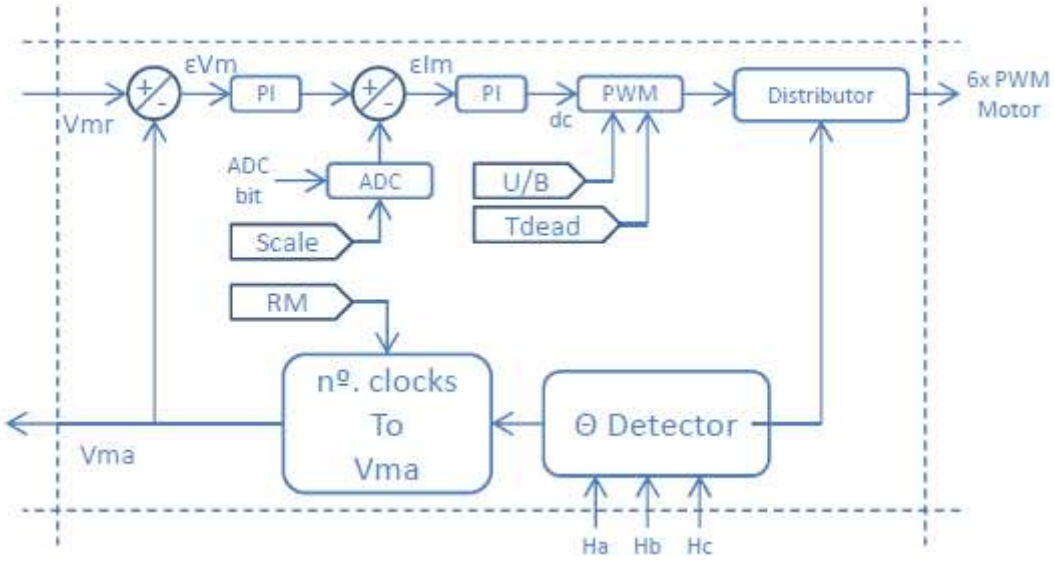


Figura 3.27 – Verilog – Motor Control – Arquitetura interna

A velocidade de referência é passada como parâmetro de entrada, sendo subtraída ao valor da velocidade atual previamente calculada com auxílio dos sensores de efeito de hall e de uma conversão do número de clocks ocorridos entre comutações dos sinais dos sensores de efeito de hall. A velocidade para além de usada para cálculo do erro de velocidade, é também um valor de saída do bloco "Motor Control". Com ambas as velocidades dos motores é possível calcular a velocidade linear e angular do robot.

Ao erro de velocidade é depois aplicado um controlador PI que, por sua vez é subtraído ao valor de corrente consumida lido com auxílio de um circuito elétrico e de um programa que converte o valor do duty cycle de uma pwm (com duty cycle proporcional ao valor de corrente consumida) neste mesmo valor de corrente atual consumida.

Por sua vez, ao erro de corrente é aplicado a um controlador PI, sendo a saída deste aplicado a um bloco de gerador de PWM como valor de duty cycle. Neste bloco, são geradas duas PWM complementares, com período e "tempo morto" configurável. Por fim, com auxílio do bloco de controlo "distribuidor", são gerados 6 sinais PWM (definidos como unipolares ou bipolares) a aplicar na ponte trifásica em H, permitindo-nos um correto funcionamento e controlo do motor.

3.3.2 Bloco Robot Control

Neste bloco é realizada toda a lógica de controlo do Robot. Os valores da velocidade linear e angular de referência do robot são parâmetro de entrada, juntamente com os valores atuais da velocidade de cada motor.

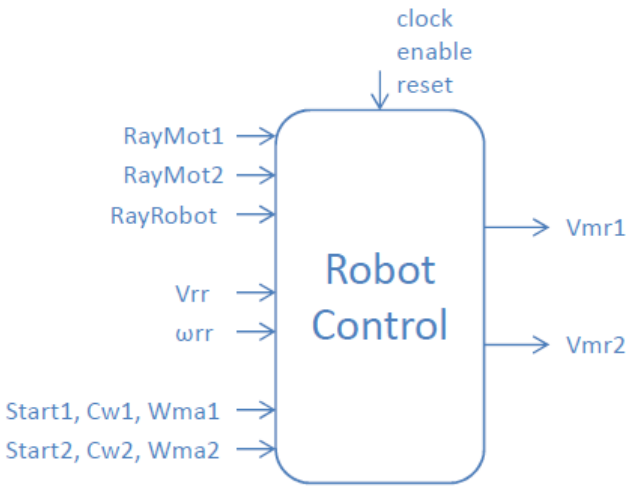


Figura 3.28 – Verilog – Bloco Robot Control

Depois de realizada uma conversão das velocidades atuais de cada motor para as velocidades linear e angular atuais do robot e da realização de uma lógica de controlo sobre as velocidades do robot, este bloco retorna o valor das velocidades referência de cada motor.

A lógica de controlo a implementar neste controlador deve privilegiar a correção do erro de velocidade linear sob a correção do erro da velocidade angular.

3.3.2.1 Conversor velocidade de motores para velocidades do robot

Este bloco converte os valores da velocidade de cada motor, na velocidade linear e angular do robot e recebe como parâmetros de entrada os raios dos motores e do robot, juntamente com o número de clocks ocorridos entre comutações de sensores de efeito de hall e o sentido de rotação de cada motor.

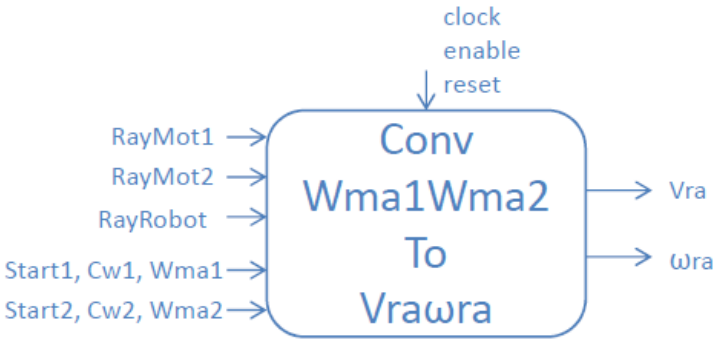


Figura 3.29 – Verilog – Bloco Conversor velocidade de motores para velocidades do robot

Sempre que ocorre um impulso em "Start1" ou "Start2", os valores internos das velocidades atuais do motor 1 ou 2 (respetivamente) são atualizados e os valores das velocidades linar e angular atuais do robot são atualizados à saída. As expressões das velocidades linear e angular do robot implementadas são as seguintes:

$$V_{robot} = \frac{1}{2} \times (V_{motor2} + V_{motor1})$$

$$\omega_{robot} = \frac{1}{2 \times R_{robot}} \times (V_{motor2} - V_{motor1})$$

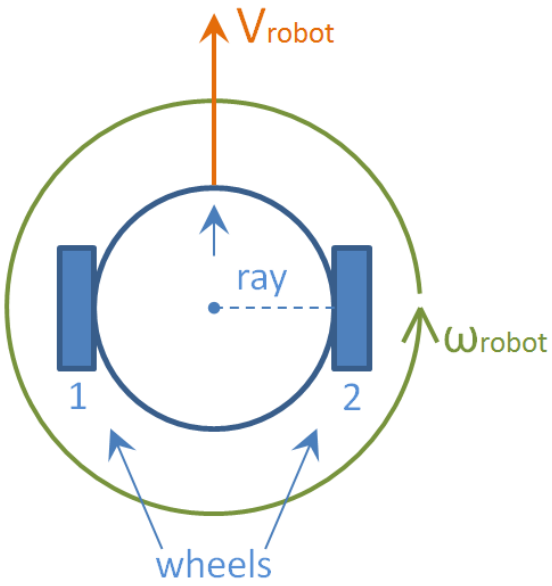


Figura 3.30 – Robot topview

3.3.2.2 Controlo Velocidade Linear Condicionada

Neste bloco é realizada uma lógica de controlo que privatiza o erro da velocidade angular ao erro da velocidade linear. Os valores passados como entrada a este bloco são as velocidades linear e angular do robot (atuais e de referência), assim como dois valores de ganho que pesam a influência do erro das velocidades linear e angular do robot.

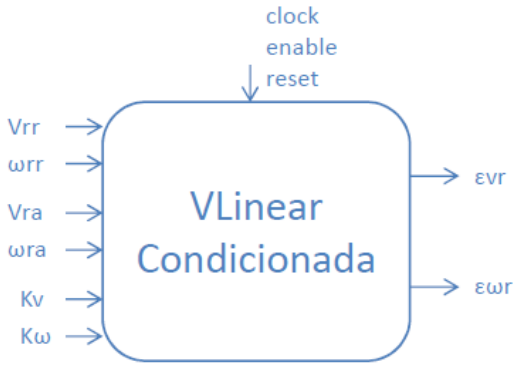


Figura 3.31 – Verilog – Bloco Controlo Velocidade Linear Condicionada

Podemos observar na figura 3.32 que o ajuste do erro da velocidade angular do robot é realizado por intermédio de um ganho K_{ω}. No entanto, o ajuste do erro da velocidade linear do robot tem um processo mais complexo. Ao valor do erro da velocidade linear do robot é aplicado um ganho inversamente proporcional ao valor absoluto do erro da velocidade angular. Quanto maior o erro da velocidade angular, menor será a influência do erro da velocidade linear no cálculo das velocidades de referência dos motores. O efeito pretendido

neste processo é que quando o robot tem um erro de velocidade angular elevado, este deverá abrandar a sua velocidade linear para corrigir o erro de velocidade angular.

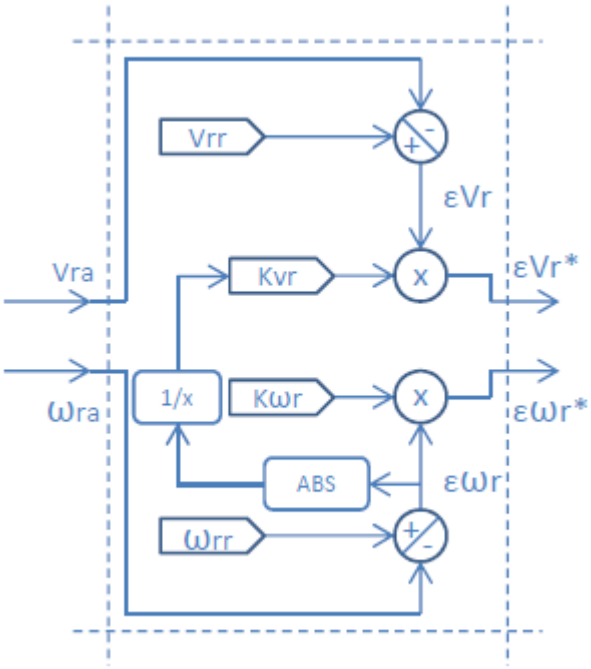


Figura 3.32 – Verilog – Bloco Controlo Velocidade Linear Condicionada, arquitetura interna

Analisando a arquitetura interna deste bloco, presente na figura anterior, podemos fazer uma análise matemática do que acontece internamente. As expressões finais para os dois erros são as seguintes:

$$\varepsilon_{\omega r}{}^{*} = K_{\omega} \times \varepsilon_{\omega r}$$

$$\varepsilon_{vr}{}^{*} = K_{v} \times \frac{1}{|\varepsilon_{\omega r}|} \times \varepsilon_{vr}$$

Uma análise mais aprofundada sobre este método de controlo será apresentada no final deste subcapítulo “Robot Control”.

3.3.2.3 Conversor εVelocidades Linear e Angular do Robot para Velocidade Referência dos Motores

Realizando um desenvolvimento das expressões matemáticas das velocidades linear e angular do robot apresentadas no sub-capítulo 3.3.2.1 (“Conversor velocidade de motores para velocidades do robot”), conseguimos chegar às seguintes expressões:

$$V_{m1} = V_{Robot} - \omega_{Robot} \times Raio_{Robot}$$

$$V_{m2} = V_{Robot} + \omega_{Robot} \times Raio_{Robot}$$

Com intuito de obter os valores de referência das velocidades de cada motor, este bloco realiza uma conversão inversa ao bloco "Conversor velocidade de motores para velocidade do robot" convertendo o erro das velocidades do robot nas velocidades de referência de cada motor.

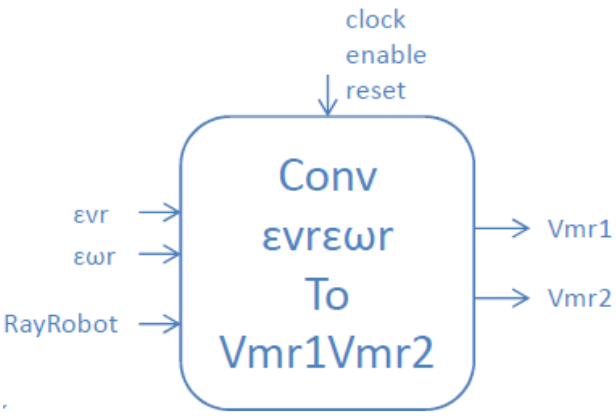


Figura 3.33 – Verilog – Bloco Conversor εVelocidades Linear e Angular do Robot para Velocidade Referência dos Motores

As equações internas a este bloco são semelhantes ao cálculo das velocidades de cada motor:

$$V_{m1r} = \varepsilon V_{Robot} - \varepsilon\omega_{Robot} \times Raio_{Robot}$$

$$V_{m2r} = \varepsilon V_{Robot} + \varepsilon\omega_{Robot} \times Raio_{Robot}$$

Por não se tratar de uma conversão direta, isto é, por se usar o erro de velocidades e não os valores das próprias velocidades, foram aplicados à saída deste bloco dois PI's que estabilizam as referências das velocidades de cada motor quando o erro das velocidades do robot forem próximos de zero. A definição dos parâmetros dos PI's devem também ser um dos passos no processo de calibração.

3.3.2.4 Robot Control – Arquitetura Interna

Na figura 3.34 podemos observar que a arquitetura interna utilizada consiste numa conversão inicial das velocidades atuais dos motores, para as suas velocidades atuais linear e angular do robot. Os erros das velocidades linear e angular do robot são pesados por um processo lógico (também apresentado na figura 3.34) e, por fim, os valores das velocidades

dos motores são adquiridos por um cálculo matemático inverso ao utilizado à entrada deste bloco.

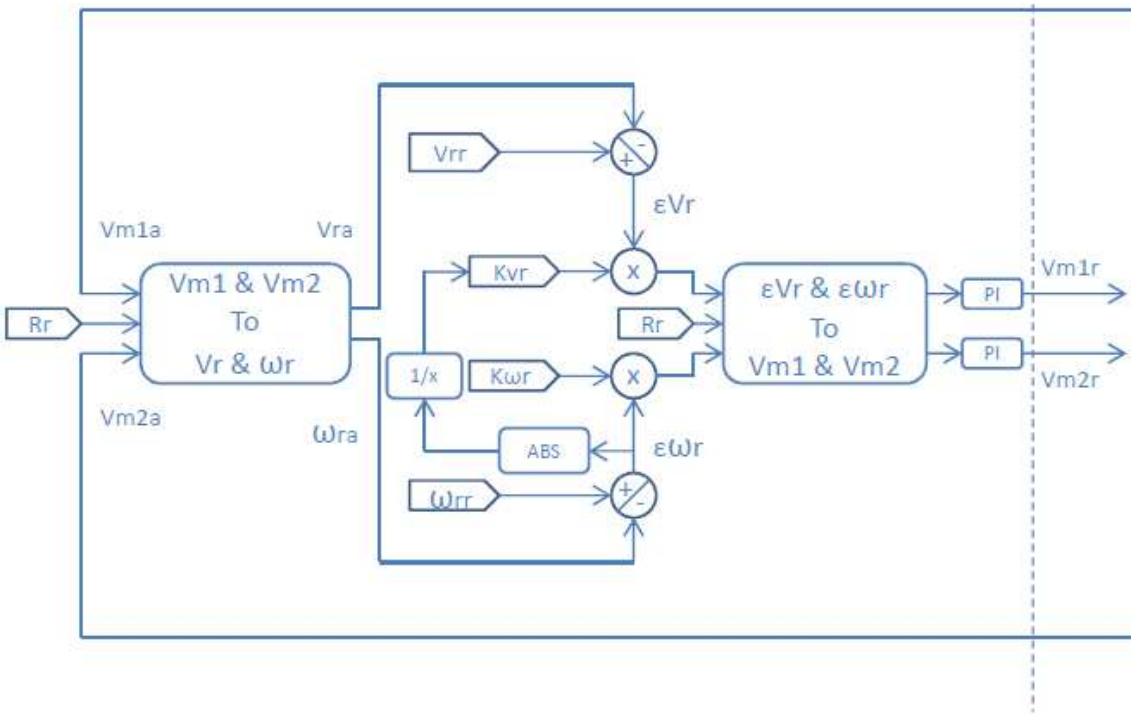


Figura 3.34 – Verilog – Robot Control – Arquitetura Interna

Foi também realizada uma análise gráfica dos valores das velocidades de cada motor, linear e angular do robot, assim como os erros de velocidades linear e angular do robot, quando submetidos a dois degraus de referência. O primeiro degrau com as referências de 10cm/s (v_{robot}) e 10rad/s (ω_{robot}), e o segundo com as referências de -5cm/s (v_{robot}) e 30rad/s (ω_{robot}). Assumiram-se os valores de 4 cm para o raio do motor (distância da roda ao centro do motor) e de 1 cm para o raio das rodas aplicadas aos motores.

Os gráficos que se seguem estão associados às velocidades de cada motor, às velocidades linear e angular do robot e dos erros das velocidades angular e linear do robot em cm/s e percentuais. Nesta primeira análise foi atribuído o valor de 10 unidades aos valores de K_v e K_ω.

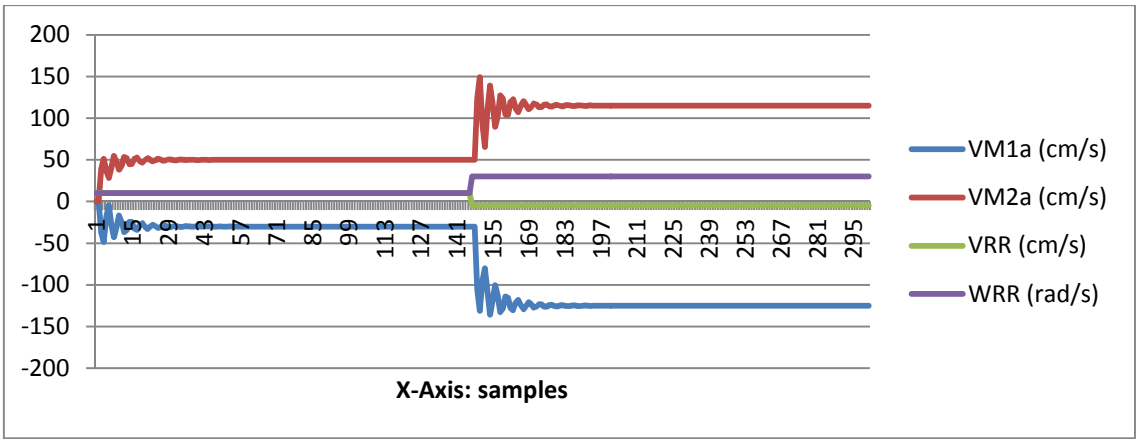


Figura 3.35 – Controlo Robot Análise 1 – Velocidades atuais dos motores

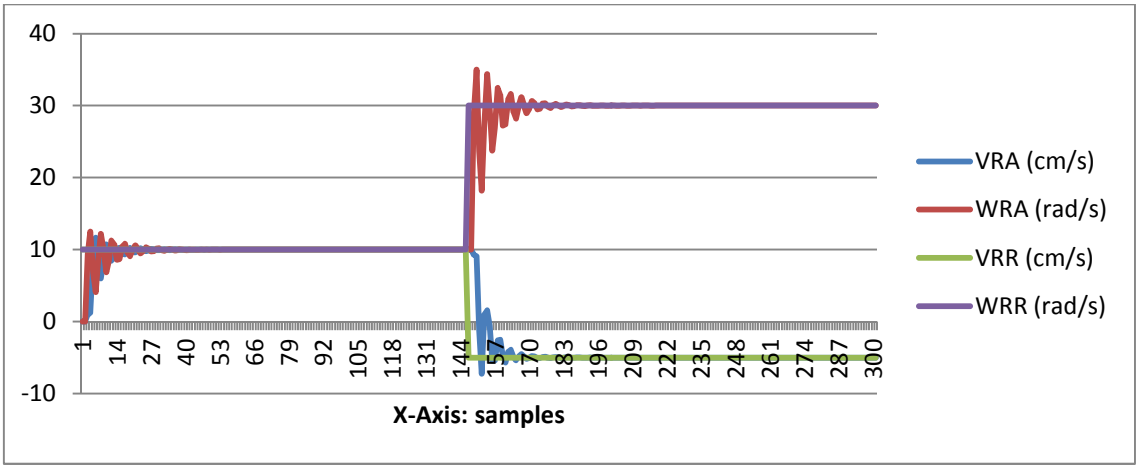


Figura 3.36 – Controlo Robot Análise 1 – Velocidades linear e angular do robot

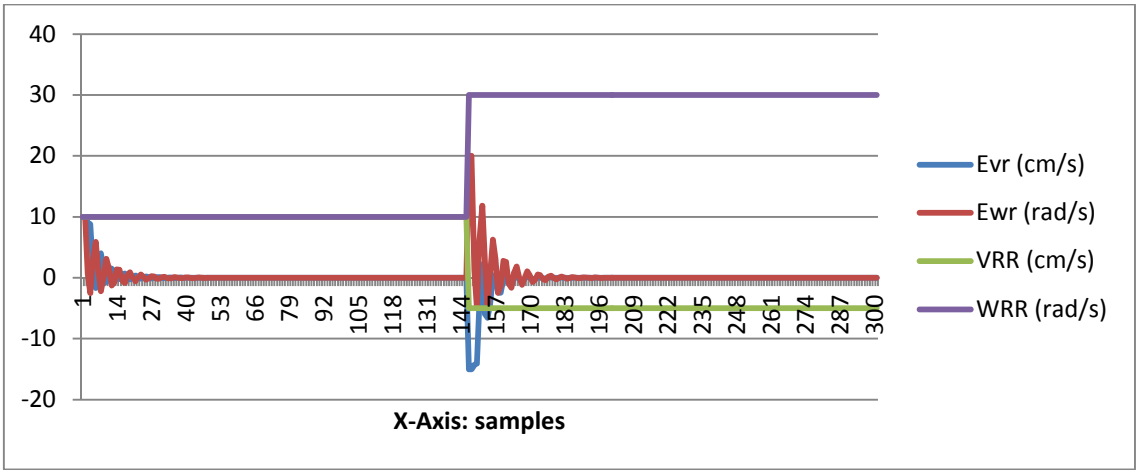


Figura 3.37 – Controlo Robot Análise 1 – Erro das velocidades linear e angular do robot.

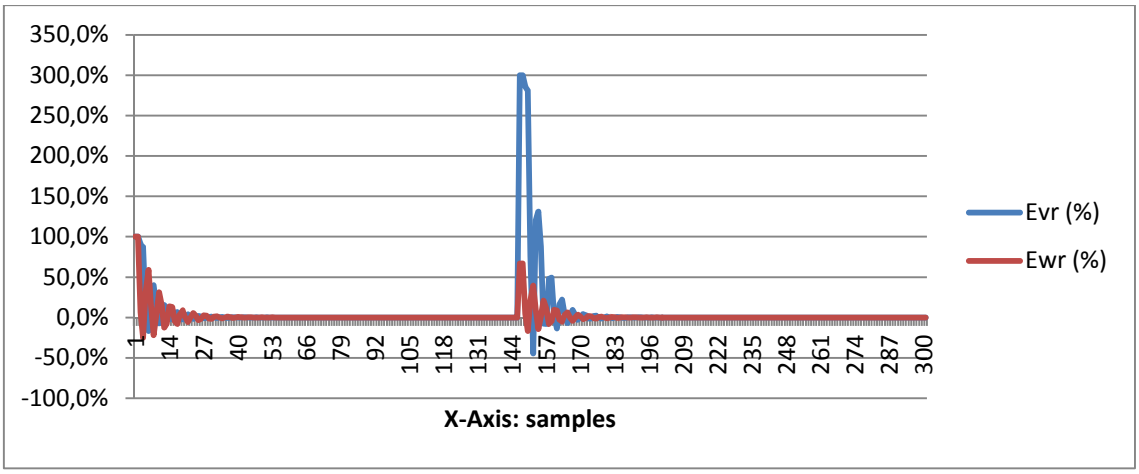


Figura 3.38 - Controlo Robot Análise 1 - Erro das velocidades linear e angular do robot.

Como podemos observar pela leitura dos gráficos nas figuras 3.36, 3.37 e 3.38, existe uma estabilização do valor das velocidades nos valores passados como referência, inicialmente em 10cm/s e 10rad/s, e de seguida em -5cm/s e 30rad/s.

Utilizando as expressões para o cálculo das velocidades dos motores, podemos também confirmar os valores das velocidades de cada motor, apresentados em gráfico:

$$V_{m1} = V_{Robot} - \omega_{Robot} \times Raio_{Robot}$$

$$V_{m2} = V_{Robot} + \omega_{Robot} \times Raio_{Robot}$$

Para o regime estacionário do primeiro degrau temos:

$$V_{m1} = 10cm/s - 10rad/s \times 4cm = -30cm/s$$
$$V_{m2} = 10cm/s + 10rad/s \times 4cm = 50cm/s$$

Para o regime estacionário do segundo degrau temos:

$$V_{m1} = -5cm/s - 30rad/s \times 4cm = -125cm/s$$
$$V_{m2} = -5cm/s + 30rad/s \times 4cm = 115cm/s$$

Observou-se também que para valores de K_ω mais baixos, consegue-se obter valores de overshooting menos intensos. Nas figuras 3.39, 3.40, 3.41, 4.42 podemos observar o desenvolvimento das velocidades de cada motor, angular e linear do robot e os erros em cm/s e percentuais das velocidades angular e linear do robot, para um novo valor k_ω (6 unidades).

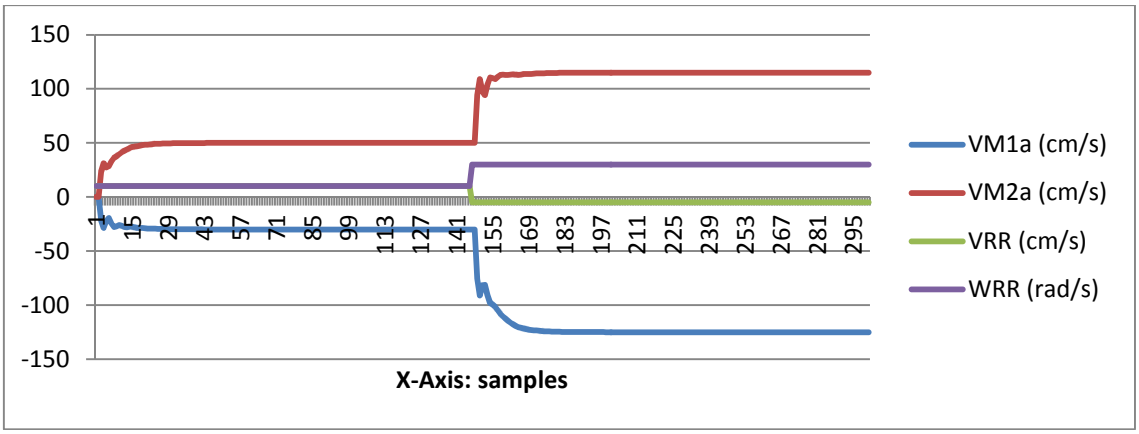


Figura 3.39 – Controlo Robot Análise 2 – Velocidades atuais dos motores.

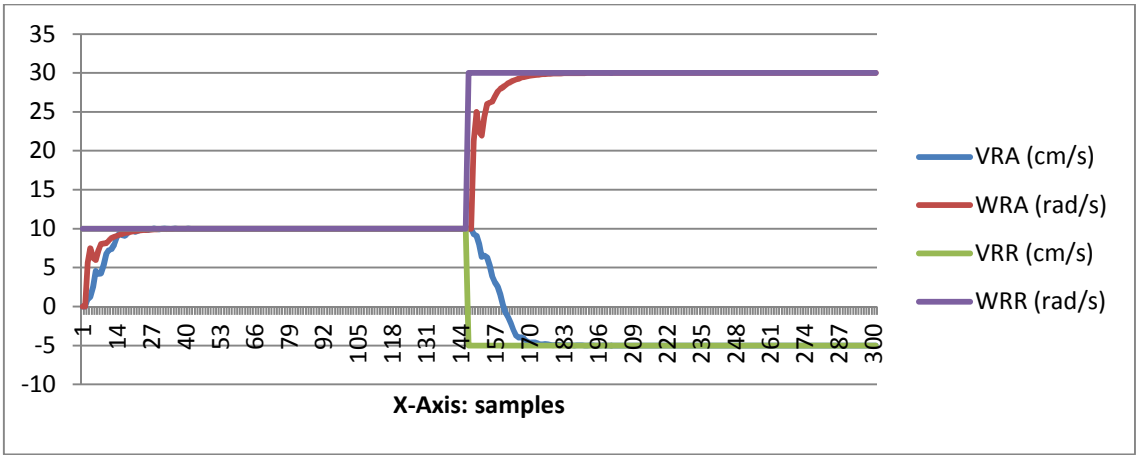


Figura 3.40 – Controlo Robot Análise 2 – Velocidades linear e angular do robot.

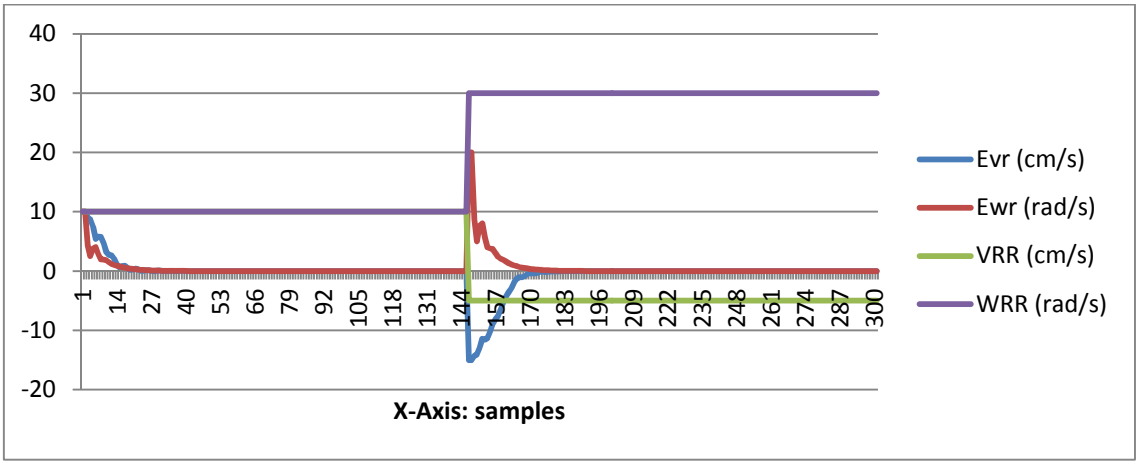


Figura 3.41 – Controlo Robot Análise 2 – Erro das velocidades linear e angular do robot (cm/s).

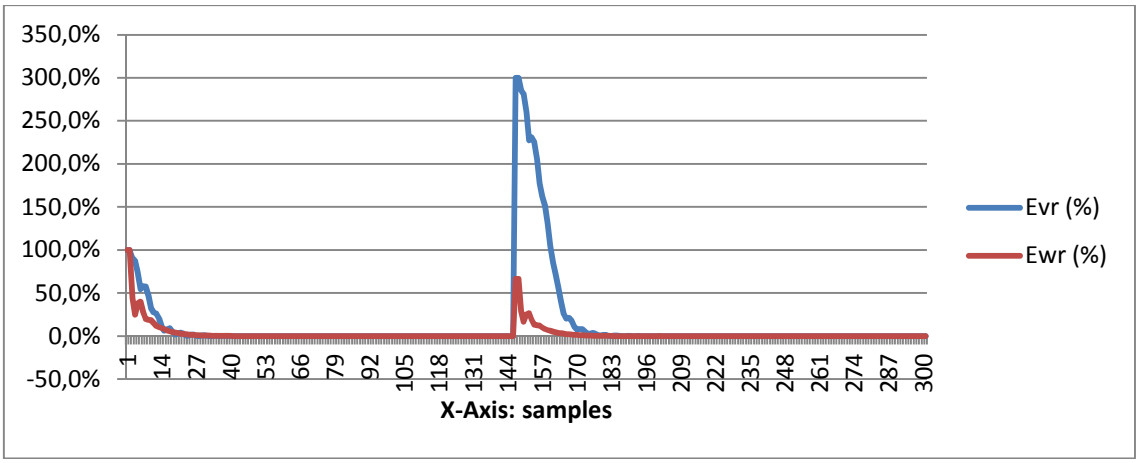


Figura 3. 42 – Controlo Robot Análise 2 – Erro das velocidades linear e angular do robot (%).

Comparando os novos valores para k_v = 10u e k_ω = 6u com os valores obtidos anteriormente para k_v = 10u e k_ω = 10u, podemos observar uma resposta mais lenta mas com overshooting reduzido.

Na figura podemos observar que o valor percentual do erro das velocidades linear e angular têm uma pequena oscilação em torno de zero. Em tabela foram registados valores entre -2,4% e 3,0% para o erro da velocidade linear e -0,1% e 0,1% para o erro da velocidade angular.

3.3.3 Comunicação porta série

A comunicação entre a FPGA e a aplicação em Lazarus é realizada com auxílio a um bloco de comunicação "Serial Port Communication" implementado em Verilog na FPGA.

Associado ao processo de receção de informação na FPGA, este bloco tem como entrada um bit "RS232_Rx" e como saídas um bit "BitReceived" e um buffer "BufferReceived". À entrada "RS232_Rx" está associada um pino físico da FPGA, recebendo sinais em RS232 que são posteriormente tratados e identificados. No final da transmissão, se esta tiver sido realizada com sucesso, o bit "BitReceived" vai a 1 e são apresentados em "BufferReceived" os valores a atribuir às variáveis internas da FPGA. Externamente a este bloco, sempre que "BitReceived" é colocado a 1, os valores internos da FPGA são atualizados com os novos valores recebidos.

Associado ao processo de envio de informação na FPGA, este bloco tem como entradas um buffer "BufferToSend", constituído com a informação a enviar da FPGA para a aplicação em Lazarus, e um bit "BitSend" que, quando colocado a 1, desencadeia o processo de envio de informação. A saída "RS232_Tx" está ligado a um pino físico da FPGA, onde é enviado o buffer em RS232. Ao valor de "BitSend" está associado um impulso com período de meio

segundo, provocando transmissão de informação da FPGA para a aplicação em Lazarus à mesma periodicidade.

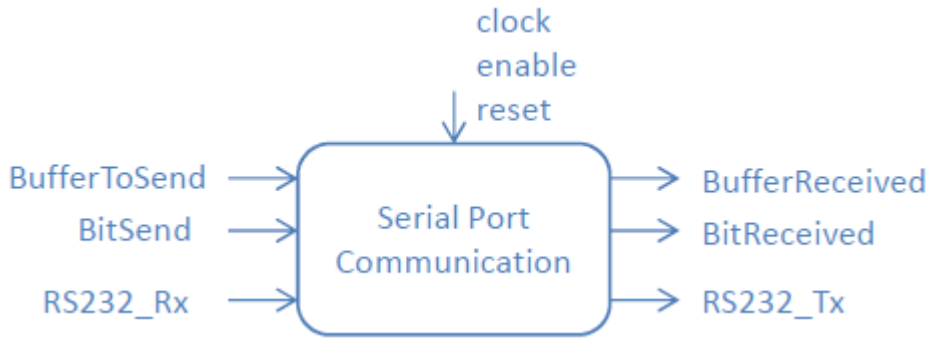


Figura 3.43 – Verilog – Bloco Comunicação porta-série.

Devido à grande quantidade de informação a enviar da aplicação em Lazarus para a FPGA e da FPGA para a aplicação em Lazarus, foi realizado um protocolo com auxílio da topologia RS232 capaz de identificar diferentes tipos de strings de informação. Assim, strings de informação enviadas da aplicação em Lazarus para a FPGA podem ser de 4 tipos, havendo apenas um tipo de string da FPGA para a aplicação em Lazarus:

- Configuração de parâmetros do motor 1 (para a FPGA) – StartByte "a" (8b) + Ganhos proporcional (16b) e integral (16b) e tempo de sampling (24b) do PI de erro de velocidade e do PI de erro de corrente/binário (x2) + tempo morto da PWM (10b) + velocidade de referência (24b) + raio da roda do motor (10b) + escala a aplicar no valor de ADC (10b) + PWM Unipolar/Bipolar (1b) + pares depolos do motor (5b) + tipo de controlo (TestMotorA, TestMotorB, RobotControl) (3b) + bit de segurança (1b) + StopByte "A" (8b);
- Configuração de parâmetros do motor 2 (para a FPGA) - StartByte "b" (8b) + Ganhos proporcional (16b) e integral (16b) e tempo de sampling (24b) do PI de erro de velocidade e do PI de erro de corrente/binário (x2) + tempo morto da PWM (10b) + velocidade de referência (24b) + raio da roda do motor (10b) + escala a aplicar no valor de ADC (10b) + PWM Unipolar/Bipolar (1b) + pares depolos do motor (5b) + tipo de controlo (TestMotorA, TestMotorB, RobotControl) (3b) + bit de segurança (1b) + StopByte "B" (8b);
- Configuração de parâmetros do robot (para a FPGA) – StartByte "i" (8b) + Ganhos proporcional (16b) e integral (16b) e tempo de sampling (24b) dos dois PI's para as referências de velocidade (x2) + Raio do robot (10b) + ganhos dos erros das velocidades linear (K_v) (10b) e angular (K_ω) (10b) + tipo de controlo (TestMotorA, TestMotorB, RobotControl) (3b) + bit segurança (1b) + "spare" (6b) + StopByte "I" (8b);

- Alteração das referências de velocidade (para a FPGA) - StartByte "r" (8b) + Vrr(24b) + Wrr(24b) + Vm1(24b) + Vm2(24b) + tipo de controlo (TestMotorA, TestMotorB, RobotControl) (3b) + bit segurança (1b) + "spare" (4b) + StopByte "R" (8b);
- Valores em FPGA (para a aplicação em Lazarus) – StartByte "s" (8b) + Velocidade (32b), Corrente consumida (32b), Odometria (32b) e sentido de rotação (1b) dos motores 1 e 2 (x2) + "spare" (6b) + StopByte "S" (8b);

A string para "configuração de parâmetros do motor 1/2" contém informação respetiva aos ganhos proporcional e integral e tempo de sampling dos PI's de erro de velocidade do motor e corrente consumida, à duração do tempo morto das pwm, à velocidade passada como referência em fase de calibração do motor 1/2, ao raio da roda aplicada ao motor 1/2, à escala aplicada ao bloco ADC para ajuste do valor lido da corrente consumida, à escolha do tipo de PWM a aplicar (Unipolar ou Bipolar), ao número de pares de pólos do motor, ao tipo de controlo a realizar pela FPGA (calibração do motor1, calibração do motor2, controlo do robot por V_{m1ref} e V_{m2ref} ou controlo do robot por $V_{robotref}$ e $\omega_{robotref}$) e ao bit de segurança que garante o correto funcionamento da comunicação com a FPGA.

O "tipo de controlo" é uma variável de 3bits enviada para a FPGA em todas as strings e distingue que ação queremos operar sobre o robot. Há 4 valores possíveis (e corretos) para esta variável, correspondentes aos modos de calibração de cada motor e aos modos de controlo do robot, por referências das velocidades de cada motor ou por referências das velocidades linear e angular do robot. Valores não definidos desta variável provocam a paragem do robot.

O "bit de segurança" é enviado a 1 em todas as strings da aplicação em Lazarus para a FPGA e, depois de confirmado o seu valor, colocado a 0. Se ao fim de três segundos este bit estiver a 0, são ajustados os valores das velocidades de referência de cada motor para a paragem do robot.

A string para "configuração do robot" contém informação respetiva aos ganhos proporcional e integral e tempo de sampling dos PI's das referências de velocidade dos motores, ao raio do robot (distância entre cada roda ao centro do robot), aos ganhos dos erros das velocidades linear (K_v) e angular (K_ω) no processo de controlo do robot, ao tipo de controlo a realizar pela FPGA e ao bit de segurança.

Por fim, a string "alteração das referências de velocidade" contém os valores de referência das velocidades de cada motor e das velocidades linear e angular do robot. Quando o controlo do robot é realizado por passagem de referência das velocidades de cada motor, os valores das velocidades de referência das velocidades linear e angular do robot são passados a zero, passando-se o inverso no caso do controlo por velocidades de referência

linear e angular do robot. No caso de ordem de paragem, as quatro velocidades serão passadas a zero.

Nesta configuração para comunicações entre a interface Lazarus e a FPGA foi utilizado um BaudRate de 115200, 8 databits, 0 bits de paridade e 1 stop bit. O tempo de envio de uma trama de 8bits pode ser calculado da seguinte forma:

$$T = \frac{Start\ bit + DataBits + Stop\ bit}{Baud} = \frac{1+8+1}{115200} \approx 87\mu s$$

Os tempos de envio das tramas completas para os casos de configuração dos Motores e Robot:

Trama “Configuração de parâmetros do motor 1/2":

[8b+(16b+16b+24b)x2+10b+24b+10b+10b+1b+5b+3b+1b+8b]=192=24Bytes

$$T = 87\mu s \times 24Bytes = 2{,}1ms$$

Trama “Configuração de parâmetros do robot":

[8b+(16b+16b+24b)x2+10b+10b+10b+3b+1b+6b+8b]=168=21Bytes

$$T = 87\mu s \times 21Bytes = 1{,}9ms$$

Trama “Alteração das referências de controlo":

[8b+24b+24b+24b+24b+3b+1b+4b+8b]=120=15Bytes

$$T = 87\mu s \times 15Bytes = 1{,}4ms$$

Trama “Alteração das referências de controlo":

[8b+(32b+32b+32b+1b)x2+6b+8b]=216=27Bytes

$$T = 87\mu s \times 27Bytes = 2{,}4ms$$

No entanto, estes valores sofrerão atrasos por influência de fatores externos, como por exemplo, o comprimento do cabo de comunicação.

Em board, foi aplicado um MAX232 para auxílio de comunicação com a porta série convertendo os níveis de tensão desta (+12V/-12V) para níveis de tensão adequados para a FPGA (+5V/0V).

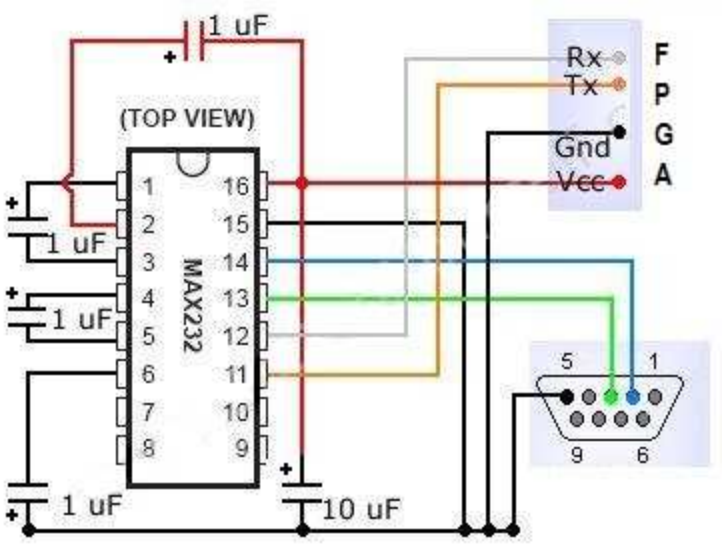


Figura 3.44 – Ligações elétricas do MAX232

Capítulo 4

Aplicação gráfica

No decorrer da realização desta dissertação, tornou-se necessária a realização de uma aplicação capaz de permitir ao utilizador não só estipular os valores em FPGA mas também, permitir ao mesmo uma análise dos valores internos calculados por ela.

Nesta aplicação devemos ser capazes de definir os valores das variáveis que influenciam o controlo dos motores e do robot. Devemos também ser capazes de analisar as velocidades linear e angular atuais do robot, assim como as velocidades e consumos de corrente atuais dos motores. Paralelamente ao processo de comando e análise, deve ser realizado um histórico de todos os eventos ocorridos, permitindo ao utilizador uma análise do decorrido anteriormente.

Para além dos dois modos de controlo do robot ("V_{robot} e ω_{robot}" ou "V_{Motor1} e V_{Motor2}"), deverá ser permitido ao utilizador a realização de uma calibração dos valores que influenciam o controlo dos motores, comandando os valores de velocidade de referência de cada motor, independentemente do resto da estrutura.

A comunicação entre a aplicação e a FPGA deverá ser realizada por intermédio da porta-série com auxílio de um protocolo em RS232.

Neste contexto, na busca de uma linguagem orientada a objetos, com fácil implementação de uma interface gráfica, e que cumpra os requisitos descritos nos parágrafos anteriores, foi realizada uma aplicação gráfica em ambiente Lazarus, com auxílio de programação em Delphi. A existência de uma vasta comunidade de utilizadores Delphi e a possibilidade de executar a aplicação em múltiplas plataformas foram também fatores decisivos para a escolha da linguagem desta aplicação.

4.1 Configuração e Calibração

A aplicação gráfica desenvolvida abre no primeiro separador. Neste, podemos aceder às configurações dos parâmetros que influenciam o controlo do robot e dos motores.

O processo de configuração geral do robot deverá seguir os seguintes passos:

1. Atribuição dos valores às variáveis de controlo do primeiro motor;
2. Confirmar o correto controlo do motor atribuindo um valor à velocidade de referência, comandando o motor num modo independente;
3. Repetir o processo definido em 1. e 2. para o segundo motor;
4. Atribuição dos valores às variáveis de controlo do robot;
5. Confirmar o controlo do robot por análise gráfica.

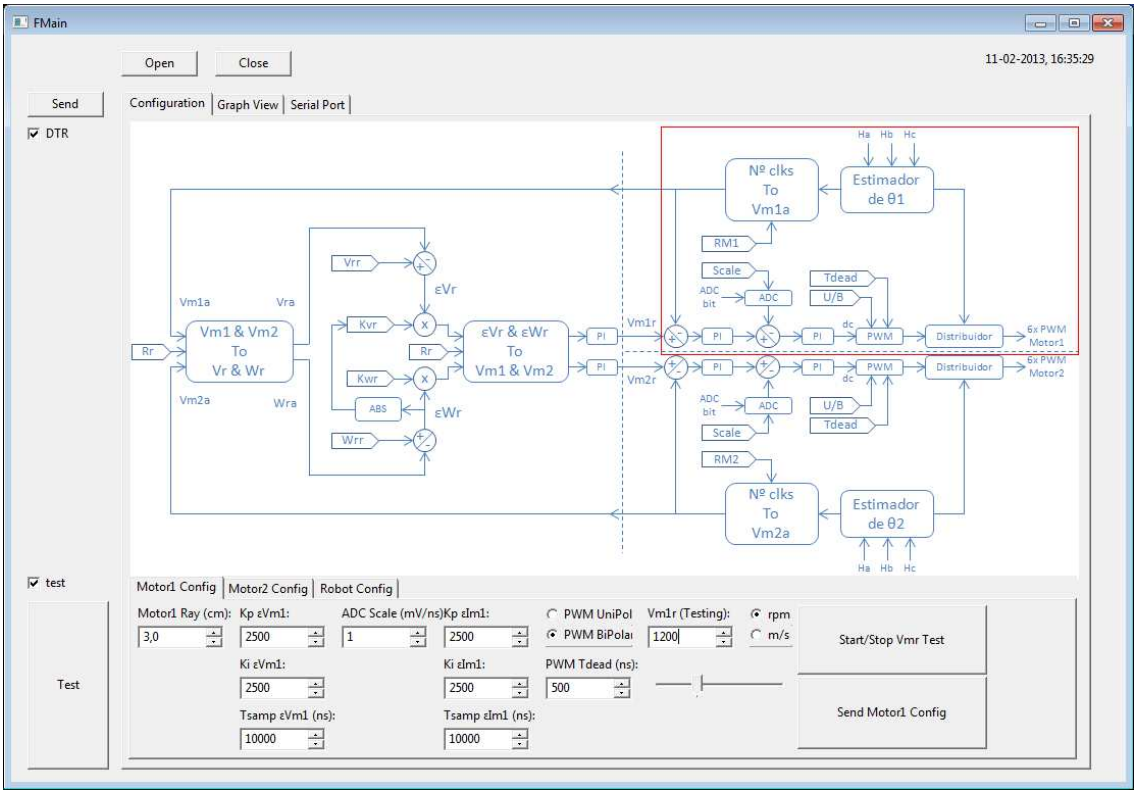


Figura 4.1 – Interface Lazarus separador1 – Configuração e Calibração dos Motores

Na figura 4.1 podemos observar o método de configuração de um dos motores. Neste modo podemos visualizar um esquemático de todos os blocos de programação, assim como as variáveis que estamos a manipular. No separador "Motor1 Config" configuram-se os valores das seguintes variáveis:

- ➔ Raio da roda do motor - "Motor1 Ray";
- ➔ Ganhos proporcional e integral e tempo de sampling do PI do erro de velocidade: "Kp εVm1", "Ki εVm1" e "Tsamp εVm1";

- ➔ Fator multiplicativo de ajuste do valor de ADC: “ADC Scale”;
- ➔ Ganhos proporcional e integral e tempo de sampling do PI do erro de corrente: “Kp εIm1”, “Ki εIm1” e “Tsamp εIm1”;
- ➔ Duração do tempo morto dos mosfet: “PWM Tdead”;
- ➔ PWM Unipolar ou Bipolar;
- ➔ Por fim, o valor da velocidade de referência a aplicar ao motor em fase de teste ou calibração, definido em rotações por minuto (“rpm”) ou metros por segundo (“m/s”).

Pressionando o botão “Send Motor1 Config” é apresentada uma mensagem de confirmação e, depois de realizada a confirmação, enviada uma trama de configuração pela porta-série, atualizando os valores em FPGA respetivos ao Motor1. Pressionando o botão “Start/Stop Vmr Test”, envia um valor de velocidade referência, iniciando também o processo de controlo (independente) do Motor1.

Sempre que são atualizados novos valores em FPGA, é realizada uma atualização no histórico da aplicação, não só com a data e a hora a que esta ocorreu, mas também quais os valores atualizados. Na figura 4.2 podemos observar um excerto do histórico respetivo a uma configuração do motor1.

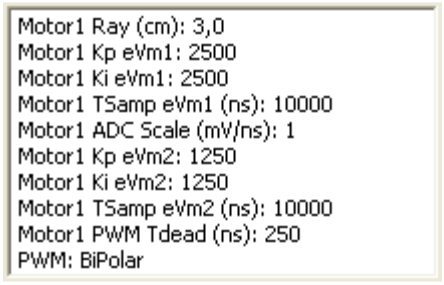


Figura 4.2 – Interface Lazarus separador1 – Histórico - Configuração Motor 1

Há semelhança do separador “Motor1 Config”, no separador “Motor2 Config” podemos configurar as variáveis respetivas ao segundo Motor.

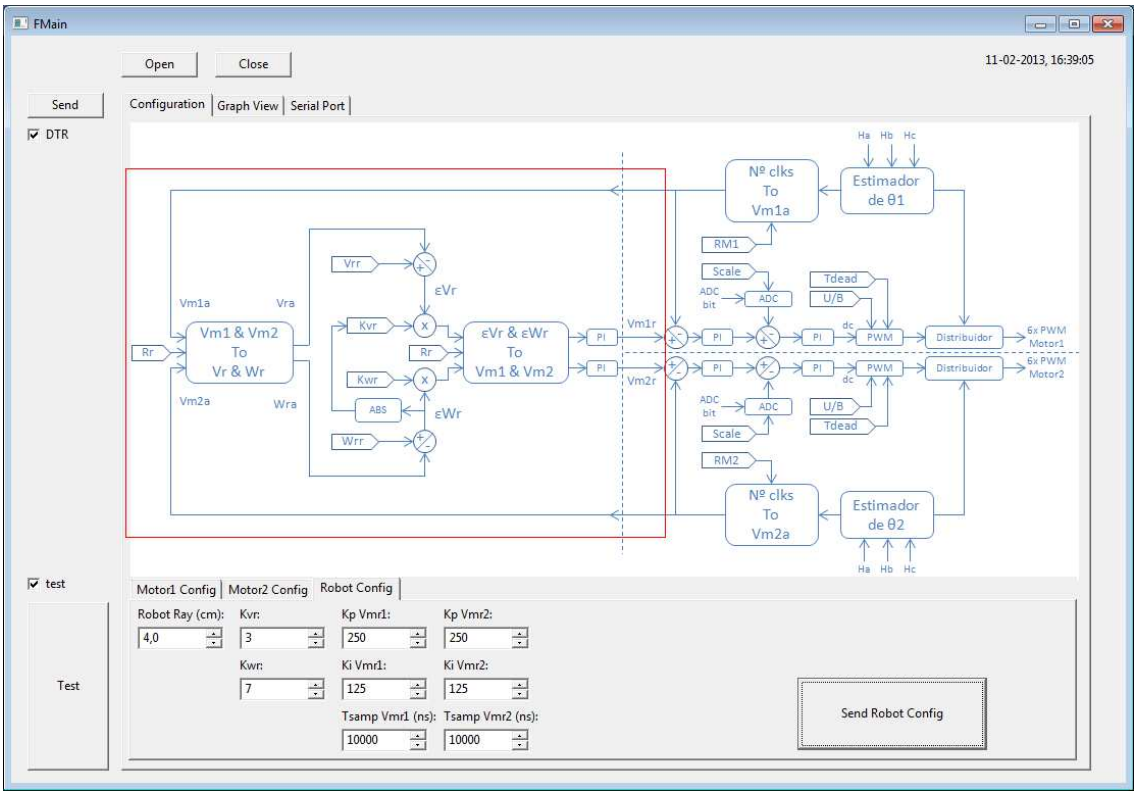


Figura 4.3 – Interface Lazarus separador1 – Configuração do Controlo do Robot

Por fim, depois de atribuídos os valores corretos aos dois motores, o processo de configuração do robot pode ser realizado.

No separador "Robot Config" podemos configurar todas as variáveis que irão influenciar o controlo geral do Robot. Uma vez mais um esquemático da arquitetura do programa realizado em FPGA é apresentado, sendo agora realçado a parte respetiva ao controlo do robot (figura 4.3). No separador "Robot Config" configuram-se os valores das seguintes variáveis:

- ➔ Raio robot (distância das rodas ao centro do robot): "Robot Ray";
- ➔ Ganhos do erro da Velocidade linear e angular: "Kv" e "Kω";
- ➔ Ganhos proporcional e integral e tempo de sampling dos PIs das velocidades ("Kp Vmr1", "Ki Vmr1", "Tsamp Vmr1", "Kp Vmr2", "Ki Vmr2" e "Tsamp Vmr2").

Tal como em "Motor1 Config" e "Motor2 Config", pressionando o botão "Send Robot Config" é apresentada uma janela de confirmação e, depois de confirmada, enviada uma trama de configuração pela porta-série, atualizando os valores em FPGA respetivos ao controlo geral do Robot. Antes de enviada, é realizada uma atualização no histórico, dos valores enviados. Na figura 4.4 podemos observar um excerto do histórico:

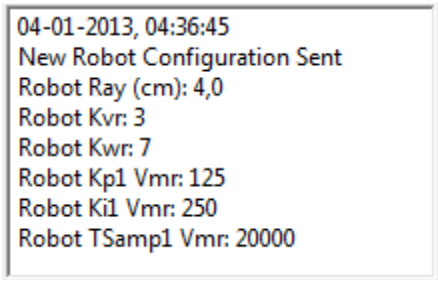


Figura 4.4 – Interface Lazarus separador1 – Histórico – Configuração Robot

No separador "Graph View", podemos observar o comportamento do robot perante as variáveis previamente configuradas. Este separador será exposto no próximo subcapítulo.

4.2 Análise Gráfica

Depois da realização das configurações dos motores e do robot, podemos fazer uma análise gráfica do desenvolvimento do valor das suas velocidades recebidos da FPGA.

No separador "Graph View" podemos observar o desenvolvimento das variáveis respetivas às velocidades e consumos dos motores e às velocidades linear e angular do robot. Clicando na "check box" ao lado dos respetivos nomes irá ativar ou desativar a visualização do desenvolvimento gráfico destas variáveis. Na figura 4.5 foram escolhidos os gráficos dos valores das velocidades dos motores e das velocidades linear e angular do robot. Estes valores são atualizados sempre que é recebida nova informação pela porta série. Os últimos valores recebidos são apresentados numericamente do lado direito. Sempre que é recebida nova informação pela porta série, é realizada uma atualização devidamente datada no histórico com os valores recebidos. No canto inferior direito da figura 4.5 podemos observar a ultima atualização realizada ao histórico.

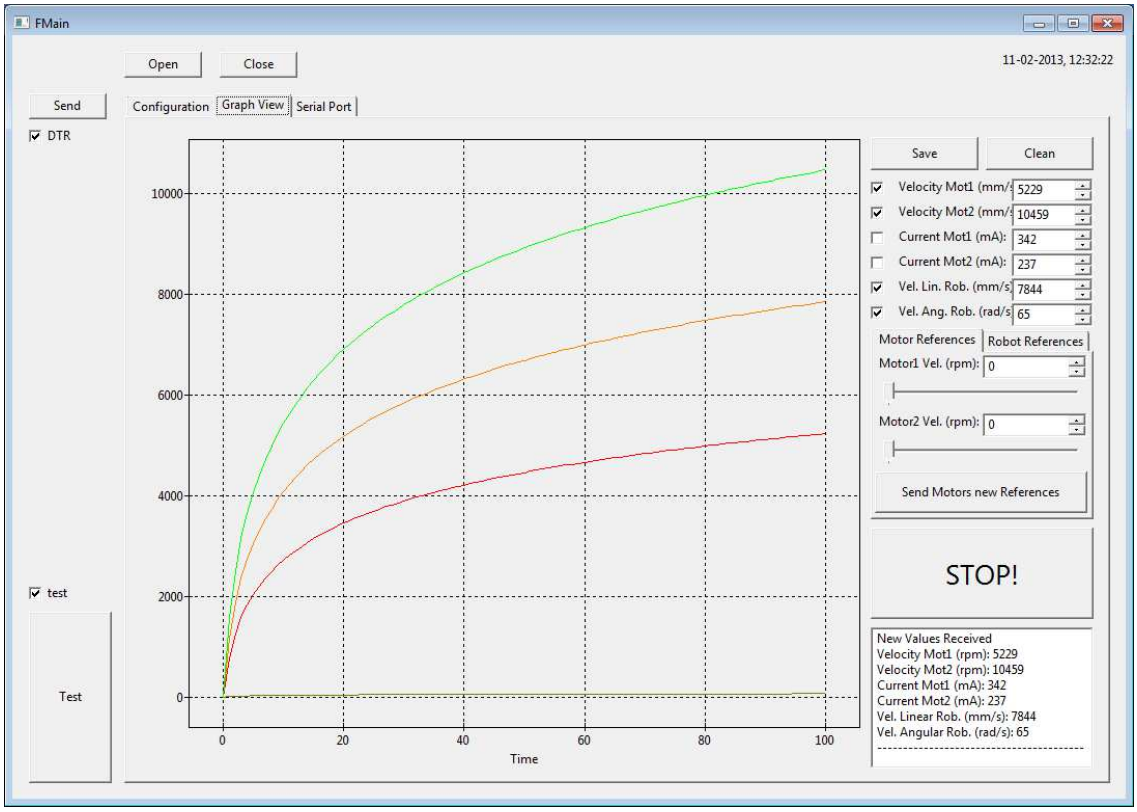


Figura 4.5 – Interface Lazarus separador2 – Interface gráfico

Tal como mencionado anteriormente, o controlo do robot pode ser realizado de duas formas: controlando as velocidades de cada motor ou as velocidades linear e angular do

robot. Na figura 4.5 podemos visualizar o método de controlo “Motor References”, onde o utilizador pode estipular os valores de referência de cada motor controlando a barra deslizante ou escrevendo diretamente o valor no campo lateral.

As novas referências são enviadas pela porta série e atualizadas em FPGA quando pressionado o botão “Send Motors New References”. Quando pressionado este botão, é também realizada uma atualização do histórico desta aplicação.

Semelhante ao controlo individual das velocidades dos motores, o método de controlo das velocidades linear e angular do robot tem o seguinte aspeto apresentado na figura 4.6.

Figura 4.6 – Interface Lazarus separador2 – Robot References

O botão de “STOP!” apresentado na figura 4.6, permite ao utilizador uma rápida ordem de paragem do robot (em caso de emergência), enviando uma trama semelhante às enviadas anteriormente, contendo esta no entanto as duas velocidades de referência a zero.

4.3 Comunicação com FPGA

No desenvolvimento da aplicação em Lazarus tornou-se necessária uma análise pormenorizada dos valores recebidos pela porta-série. Assim, para análise e depuração (debugging) das tramas enviadas e recebidas pela porta-série, foi acrescentado um terceiro separador onde podemos observar pormenorizadamente toda a informação recebida e enviada pela porta-série.

Citando o capítulo referente ao programa em Verilog da comunicação pela porta-série, as tramas enviadas e recebidas da FPGA são constituídas por:

- ➔ Enviadas (pela FPGA):
  - "Start Byte" e "Stop Byte" – no início e final da trama, utilizados para identificação da informação recebida;
  - "clock counter 1" e "clock counter 2" – número de clocks ocorridos entre cada comutação dos sensores de efeito de hall do motor 1 e 2;
  - "adc counter 1" e "adc counter 2" – número de clocks proporcional ao DutyCycle da pwm aplicada ao pino da FPGA associado à leitura em R_{sense} da corrente consumida – bloco Verilog "ADC".
- ➔ Recebidas (pela FPGA) :
  - "Start Byte" e "Stop Byte" – no início e final da trama, utilizados para identificação da informação recebida;
  - Ganhos proporcional e integral e tempo de sampling dos 6 PIs - 2 em cada motor (erro velocidade e erro de corrente) e 2 à saída do controlo do robot (referencia de velocidade de cada motor);
  - Raio das rodas aplicadas aos motores e do robot;
  - Escala do ADC, PWM Unipolar/Bipolar, duração do tempo morto das pwm;
  - Velocidades de referência – Linear e Angular do Robot ou de cada Motor;

Toda a informação recebida pela aplicação deve ser tratada antes de ser apresentada como correntes e velocidades dos motores e do robot. As expressões para o cálculo das velocidades dos motores são as seguintes:

$$V_{motor1}(rpm) = \frac{1}{"clock\ counter\ 1"} \times \frac{1}{f_{clock}} \times \frac{60°}{360°} \times \frac{1}{p} \times 60seg = \\ = 10 \times \frac{1}{"clock\ counter\ 1"} \times \frac{1}{f_{clock}} \times \frac{1}{p}$$

Com p sendo o número de pares de pólos do motor e f_{clock} a frequência do clock da FPGA.

Para uma frequência de clock = 50MHz, e 1 par de pólos, temos:

$$V_{motor1}(rpm) = \frac{5 \times 10^8}{"clock\ counter\ 1"}$$

$$V_{motor1}(cm/s) = V_{motor2}(rpm) \times (2 \times \pi \times Raio_{motor1})$$

Com o valor do raio da roda aplicado ao motor2 definido em centímetros.

De um método semelhante, para o motor 2 temos:

$$V_{motor2}(rpm) = \frac{5 \times 10^8}{"clock\ counter\ 2"}$$

$$V_{motor2}(cm/s) = V_{motor2}(rpm) \times (2 \times \pi \times Raio_{motor2})$$

Com o valor do raio da roda aplicado ao motor2 definido em centímetros.

O cálculo para as velocidades linear e angular do robot é o seguinte:

$$V_{Robot}(cm/s) = \frac{1}{2} \times \left(V_{motor1}(cm/s) + V_{motor2}(cm/s)\right)$$

$$\omega_{Robot}(rad/s) = \frac{1}{2 \times Raio_{Robot}} \times \left(V_{motor1}(cm/s) + V_{motor2}(cm/s)\right)$$

Com o valor do raio do robot definido em centímetros.

Por outro lado, a trama enviada pela aplicação em Lazarus pela porta-série é também tratada.

Todo este processo de recepção/envio, segmentação, cálculo e desenho/apresentação gráfica pode ser monitorizado “passo-a-passo” no separador “Serial Port”.

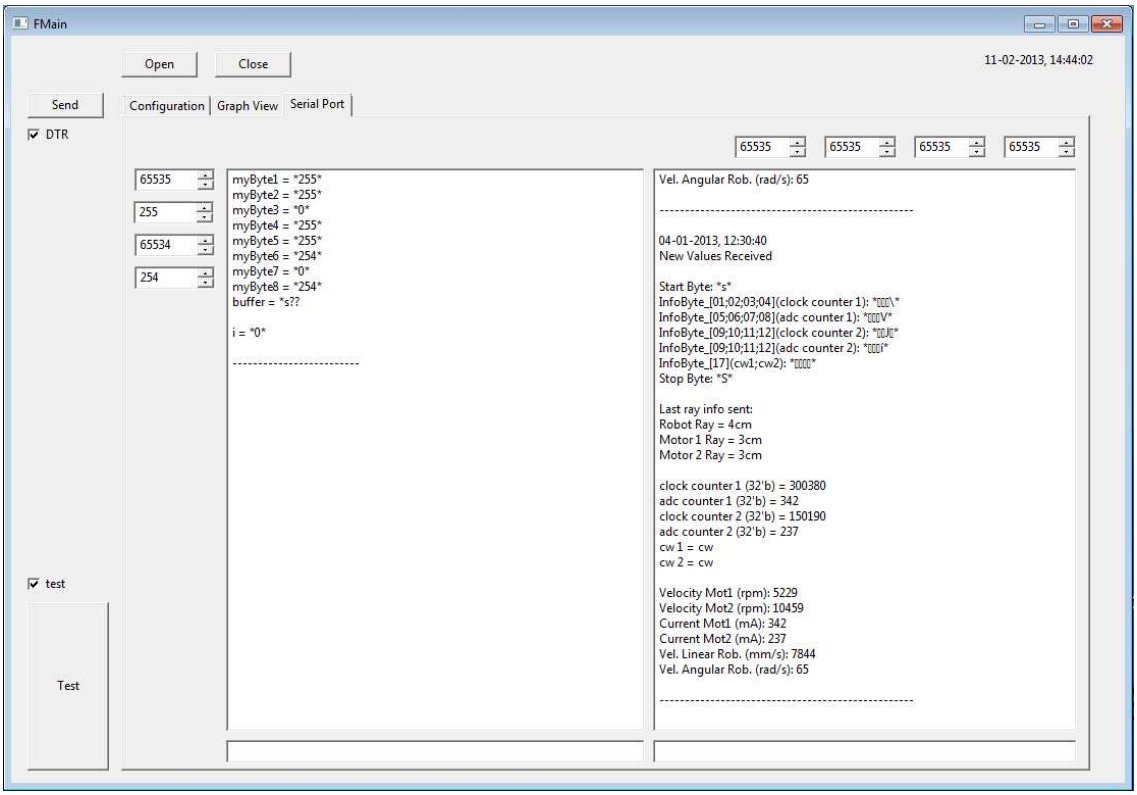


Figura 4.7 – Interface Lazarus separador1 – comunicações porta-serie

Na figura 4.7 podemos observar um log do que está a ser enviado (lado esquerdo), e do que está a ser recebido assim como os vários passos de tratamento da trama recebida (lado direito).

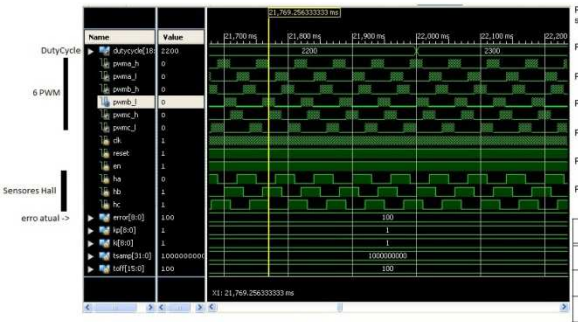
Capítulo 5

Experimentação e Resultados

5.1 Simulação de Programas em Verilog

Ao longo do desenvolvimento da programação em Verilog, foi necessário confirmar o comportamento dos vários programas realizados.

Paralelamente com o programa de escrita Verilog "ISE" da Xilinx foi utilizado o programa"ISim" também da Xilinx para simulações de possíveis cenários.Com este programa "ISim", foi simulado o sinal de clock, dos sensores de feito de Hall, assim como outros valores internos da FPGA, verificando-se que os resultados obtidos eram os esperados.



Hall Sensor A	Hall Sensor B	Hall Sensor C	Phase A	Phase B	Phase C
1	0	0	$-V_{DCB}$	$+V_{DCB}$	NC
1	0	1	NC	$+V_{DCB}$	$-V_{DCB}$
0	0	1	$+V_{DCB}$	NC	$-V_{DCB}$
0	1	1	$+V_{DCB}$	$-V_{DCB}$	NC
0	1	0	NC	$-V_{DCB}$	$+V_{DCB}$
1	1	0	$-V_{DCB}$	NC	$+V_{DCB}$

Figura 5.1 – ISim – Simulação dos sinais de comando da ponte trifásica em H

Na figura 5.1 podemos observar o resultado de uma simulação do bloco "motor control" com duty cycle da pwm constante, onde são variados os valores de sensores de hall e obtidos os 6 sinais pwm a aplicar na ponte H.

A análise das simulações realizadas leva-nos a concluir se o programa realizado cumpre os requisitos definidos ou não. Grande parte destas simulações estão apresentadas no capítulo 3.

5.2 Experimentação do Driver de Potência e do Motor

O motor utilizado em laboratório para testes nesta dissertação foi um Maxon EC flat com sensores de efeito Hall. Este motor é alimentado a 9V, tendo uma velocidade de 7990 min^{-1} sem carga e consumo de corrente de 145mA. Os valores de velocidade e corrente consumida nominais são de 4830 min^{-1} e 1,96A. Este Motor possui também 8 pares de polos.

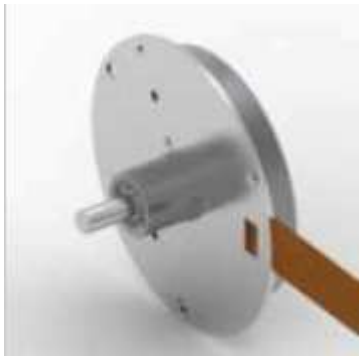


Figura 5.2 – Motor Maxon brushless EC flat with Hall sensors (part number: 200188)

Foi realizada uma proposta de orçamento inicial para implementação do driver de potência, no entanto, devido ao seu elevado valor e à disponibilidade de outros drivers de potência em laboratório, foram utilizados drivers com o seguinte esquemático:

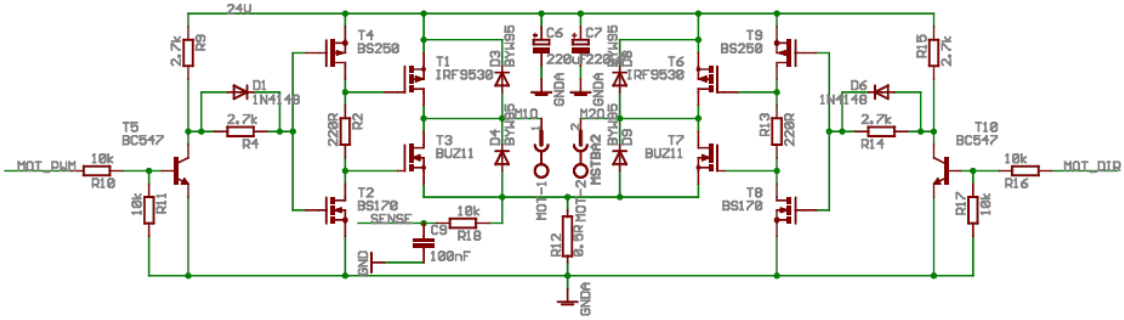


Figura 5.3 – Driver de potência utilizado em laboratório

No esquemático podemos observar dois dos três braços da ponte H pretendida, pelo que, para a implementação do driver de potência a alimentar o motor, foram utilizadas duas das placas apresentadas na figura 5.3 (3 dos 4 braços disponíveis).

Em laboratório a resistência R_{sense} (R_{12} – entre a terra e as Sources dos Mosfet T3 e T7) foi curto-circuitada e aplicada uma nova resistência R_{sense} entre a massa e as Sources dos Mosfet dos três braços. O nível de tensão lido nesta nova resistência é proporcional à corrente consumida no driver (desprezando perdas, tipicamente baixas em motores Brushless). Esta tensão é filtrada com um filtro passa-baixo RC, e aplicada ao circuito ADC mencionado no sub capítulo 3.3.1.5 (“ADC”).

A aplicação destes drivers de potência condiciona o controlo dos mosfets, na medida em que os sinais aplicados aos mosfets num mesmo braço estão eletronicamente associados, sendo o tempo morto provocado por intermédio de um atraso implementado por um circuito eletrónico e não o configurado na FPGA. Assim, dos 6 sinais pwm de saída da FPGA, apenas os 3 sinais pwm (“High”) deverão ser aplicados à ponte H (um a cada braço). Apesar de apenas 3 sinais pwm de saída serem aplicados à ponte H, é possível visualizar as formas de onda dos 6 sinais nos pinos da FPGA em osciloscópio. Na figura 5.4 podemos visualizar as formas de onda de dois sinais de comando a aplicar aos dois mosfets de um mesmo braço da ponte H, à esquerda com pwm Bipolar e à direta com pwm Unipolar.

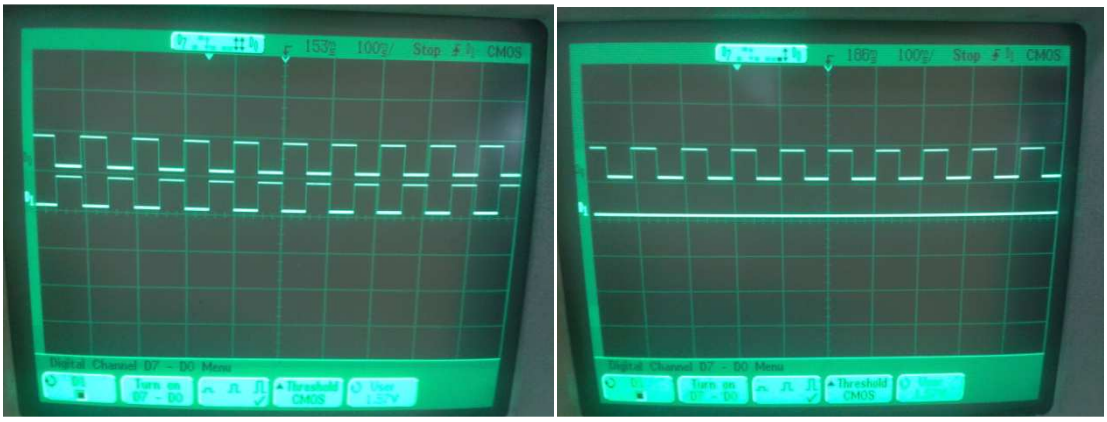


Figura 5.4 – Osciloscópio – PWM Bipolar (esquerda) e Unipolar (direita) para um braço da Ponte H – DC = 50%

Na figura 5.4 podemos observar que a pwm aplicada tem duty cycle de 50%. Este valor foi definido em fase de testes através da aplicação gráfica desenvolvida.

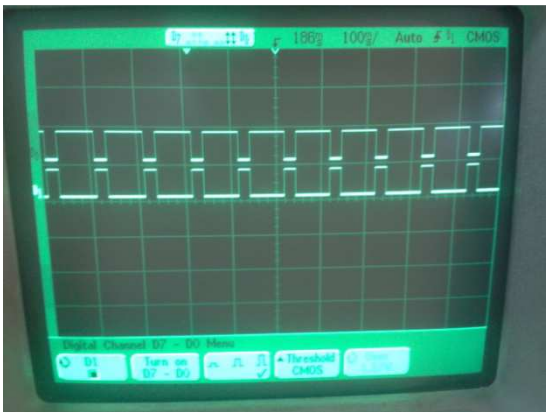


Figura 5.5 – Driver Osciloscópio – PWM Bipolar para um braço da Ponte H – DC = 75%

Na figura 5.5 podemos observar uma pwm com duty cycle de 75%, também definido em fase de testes na aplicação gráfica.

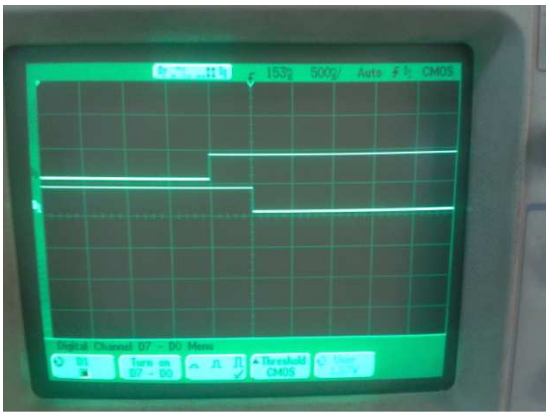


Figura 5.6 – Osciloscópio – PWM Bipolar para um braço da Ponte H – Tempo Morto = 500ns

Foi também comprovado que o valor para o tempo morto lido em osciloscópio coincidia com o valor estipulado. Na figura 5.6 podemos observar os sinais de comando de um braço da ponte H, assim como a duração do tempo morto de 500ns.

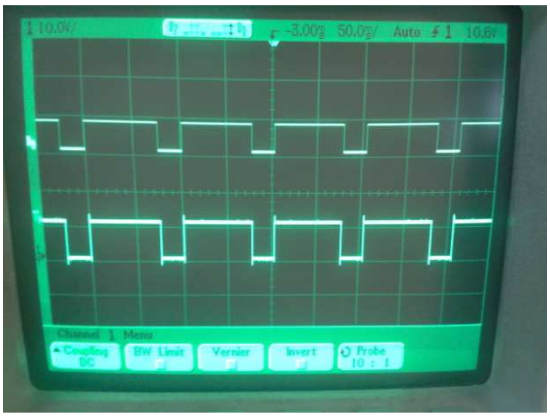


Figura 5.7 – Osciloscópio – sinal de comando de um braço da ponte H e respetivo valor à saída.

Na figura 5.7 podemos observar a forma de onda na saída de um dos braços da ponte H, assim como o sinal de comando aplicado a esse braço.

Note-se que, o comando aplicado no driver é na verdade o comando do mosfet superior, sendo o seu complemento aplicado ao mosfet inferior (a menos de um atraso de "tempo morto" implementado em hardware na placa). Os níveis máximos e mínimos lidos nesta pwm, correspondem à alimentação do driver (10V, menos a queda de tensão no mosfet superior) e a uma tensão próxima de ground (0V, mais a queda de tensão no mosfet inferior e a queda de tensão na resistência Rsense).

Depois de testada a parte de potência, e configurados os parâmetros de controlo do motor através da aplicação gráfica realizada, foram realizados testes aos processos de leitura e controlo de velocidade do motor.

Inicialmente foi realizado um controlo de malha fechada apenas com feedback da velocidade do motor atual, tendo sido limitada a corrente na fonte de alimentação.

Na figura 5.8 podemos observar uma imagem retirada da aplicação gráfica desenvolvida que demonstra o desenvolvimento das velocidades do motor como resposta a várias referências atribuídas.

Figura 5.8 – Aplicação gráfica – Desenvolvimento da velocidade do motor (mm/s).

As referências de velocidade atribuídas foram as seguintes:

- ≈20 seg: 40m/s;
- ≈30 seg: 10m/s;
- ≈35 seg: -20m/s;
- ≈43 seg: -40m/s;
- ≈60 seg: 20m/s;
- ≈90 seg: -25m/s;

Pela análise da figura 5.8 podemos afirmar que, apesar de alguma instabilidade para velocidades negativas (sentido contra ponteiros do relógio), o controlo por malha fechada com feedback de velocidade demonstrou-se eficaz.

Paralelamente ao controlo foram também adquiridos os sinais aplicados à ponte trifásica H, assim como as formas de onda aplicadas ao motor.

Na figura 5.9 podemos observar os 3 sinais digitais de saída da FPGA aplicadas aos braços da ponte H, assim como os sinais de saída de dois braços da ponte trifásica H.

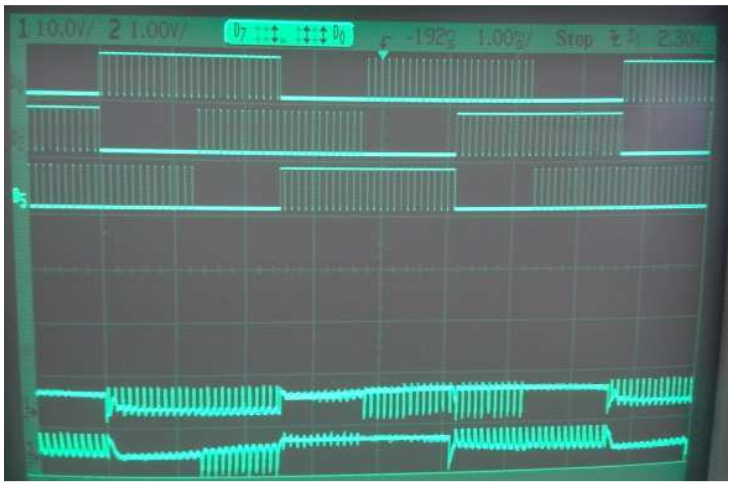


Figura 5.9 – Osciloscópio – sinais de comando pwm A, B e C + sinais de saída (braços A e B)

Por fim, foi acrescentado um feedback da corrente consumida pelo motor à malha anterior para controlo de consumo de corrente e proteção ao motor, no entanto, devido à dificuldade de calibração e ajuste dos valores do controlador PI de corrente, não foi possível realizar um controlo com feedback de corrente do motor com sucesso.

Capítulo 6

Conclusões e trabalhos futuros

6.1 Principais Conclusões do trabalho

Por análise dos gráficos apresentados no capítulo 3, respetivos ao método de controlo "Velocidade Linear Condicionada", este método mostrou ser eficaz para uma rápida correção da velocidade angular do robot. Observou-se também ser possível influenciar as respostas transitórias por alteração dos valores de k_v e k_ω.

A aplicação gráfica implementada demonstrou-se capaz de configurar e alterar os valores internos da FPGA, influenciando o método de controlo do robot e dos motores, assim como visualizar os valores da velocidade do motor.

A nível de hardware, foi possível visualizar as formas de onda de comando da ponte trifásica em H (6 pinos de saída da FPGA), assim como a saída da ponte trifásica em H a aplicar ao motor. Tendo-se também conseguido controlar a velocidade do motor brushless com malha fechada por feedback de velocidade. No entanto, devido à dificuldade de configuração do controlador PI de corrente, acrescentando um feedback de corrente consumida para controlo de consumo e de proteção do motor, o sistema demonstrou-se instável.

6.2 Sugestões para Trabalhos Futuros

Desde o início desta dissertação que a continuidade da evolução deste tema esteve em conta, não só a nível de análise de novos tipos de controlo (ex: Controlo por "Velocidade Linear Condicionada"), como também para aplicação em diferentes estruturas mecânicas em robótica móvel (ex: Wheelchair).

Para além destas novas perspetivas mencionadas no parágrafo anterior, novos objetivos e tarefas foram surgindo tornando-se potenciais motivadoras de projetos futuros.

Cito de seguida algumas dessas sugestões:

Aplicação de controlos para vários tipos de motores e respetivos drivers de potência: a robótica móvel utiliza uma grande variedade de motores, cada um com características mais específicas (ex: expressões matemáticas da velocidade ou binário), ou mais generalizadas (ex: sensores de posicionamento); Com informação suficiente no método "configuração/calibração" na interface Lazarus, este pode ser utilizado para testes de qualquer tipo de motores, mesmo não estando estes associados a um robot.

Melhorar o sistema de calibração dos controladores PI: devido à grande variedade de estruturas robóticas, seria interessante aplicar um sistema auto-configurável capaz de adaptar os parâmetros dos controladores PI ao sistema implementado, permitindo no entanto ao utilizador uma certa flexibilidade para alteração desses mesmos valores, e estudo das respostas do sistema a essas mesmas alterações.

Alteração do número de motores e disposição dos mesmos no robot: existe uma grande variedade de topologias de robots de condução autónoma, não só no número de rodas, como na disposição das mesmas; A evolução deste projeto com intuito de identificação e controlo dessa variedade de topologias, estimularia a utilização do resultado final deste projeto em várias aplicações não só académicas (ex: concursos ou festivais) como também em aplicações do dia-a-dia (ex: otimização de percursos percorridos em veículos fabris).

Otimização do tempo de resposta do código já realizado: Um dos processos mais condicionante a nível temporal na programação desenvolvida nesta dissertação é o protocolo de comunicação pela porta-série em RS232. Sugere-se a implementação de diferentes tipos de protocolo de comunicação. Para além da comunicação, sugere-se também uma busca dos melhores valores a aplicar nos controladores PI (ganhos "auto-ajustáveis") para controlo de velocidade dos motores. A nível de cálculo do valor de variáveis internas na FPGA sugere-se também uma otimização do cálculo da divisão.

Revisão do Hardware utilizado: a organização da arquitetura dos drivers de potência, reduzirá as dimensões das placas de circuito impresso. A utilização de seis mosfets "independentes" em vez de 3 pares de mosfets (3 braços) numa ponte H, permitiria tirar proveito do controlo do tempo morto das pwm.

Implementação de novos métodos de controlo do Robot: a nível académico, a aplicação de novos métodos de controlo do robot permitiria a realização de novos estudos e análises de otimização.

Desenvolvimento de nova aplicação gráfica e novos protocolos de comunicação: A aplicação em ambiente fabril deste tipo de controlo pode limitar os utilizadores a um posto fixo indesejado; O desenvolvimento de novos protocolos de comunicação iria facilitar o acesso ao controlo dos motores. Sugere-se uma aplicação em Android, com comunicação USB ou Wireless.

Referências

[1] V. Sornam Viswanathan, *Embebedded Control Using FPGA*, 2005

[2] National Instruments: http://www.ni.com. FPGA based control: Millions of transistors at your command, 2012

[3] Stephen Brown. FPGA and CPLD architectures: A Tutorial. IEEE Design And Test Of Computers, 1996

[4] Charles H Roth Jr. Digital System Desgin Using VHDL. Brooks/Cole,1998

[5] Xilinx. Comparing and Contrasting FPGA and Microprocessor System Design and Development, july 2004.

[6] M. H. Westbrook, *The Electric Car: Development and Future of Battery, Hybrid and Fuel-Cell Cars* vol. 38, 2001.

[7] M. Zeraouila, et al., "Electric motor drive selection issues for HEV propulsion systems: a comparative study," ins Vehicle Power and Propulsion, 2005 IEEE Conference, 2005

[8] http://paginas.fe.up.pt/~ajm/www_sam/mcc_sam.pdf - 3 Fev 2013

[9] http://www.abb.pt/product/seitp322/9e80e9093dc5bc26832575a6006b87a1.aspx - 3 Fev 2013

[10] http://paginas.fe.up.pt/~ee02060/MIT.html - 3 Fev 2013

[11] M. H. Westbrook, The Electric Car: Development and Future of Battery, Hybrid and Fuel-Cell Cars vol. 38, 2001.

[12] www.delphibasics.co.uk/ - 3 Fev 2013

[13] Application note – AN10661 – Brushless DC motor control using LPC2141

[14]F. A. P. Lopes, "Estudo e Comparação de Diferentes Métodos de Controlo de Motores Síncronos com Ímanes Permanentes," FEUP, 2008.

[15]. R. Krishnan, Electric motor drives: modeling, analysis, and control, 2001.

[16]. Jaroslav Lepka, Petr Stekl, "3 Phase AC Induction Motor Vectorl Control Using a 56F80x, 56F8100 or 56F8300 Device", Freescale Semiconductor, 2005

[17]http://www.oho-elektronik.de/ - 3 Fev 2013

[18]http://www.oho-elektronik.de/pics/UM_GODIL.pdf - 3 Fev 2013

[19] http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf - 3 de Fev 2013

[20] http://www.asic-world.com/verilog/veritut.html - 3 Fev 2013

[21] Shabiul Islam, Mukter Zaman, Bakri Madon, and Masuri Othman, *Designing Fuzzy Based Mobile Robot Controller using VHDL*, Issue 1, Volume 2, 2008

[22]Glen Young, product marketing manager, Actel Corporation, *Motion Control and Mixed-signal FPGAs*

[23]Stéphane Simard, Jean-Gabriel Mailloux, and Rachd Beguenane, *Prototyping Advanced Control Systems on FPGA*, Volume 2009, Article ID 897023

[24] Daijin Kim, *An Implementation of Fuzzy Logic Controller on the Reconfigurable FPGA System*, IEEE Transations on Industrial Electronics, Vol.47, No.3, June 2000

[25] Macon motor ag / Marcel Honegger, *CANopen for small drives*

[26] Wei Zhao, Byung Hwa Kim, Amy C. Larson, and Richard M. Voyles, *FPGA Implementation of Closed-Loop Control System for Small-Scale Robot*

[27] O. Al-Ayasrah, T. Alukaidey, G. Pissanidis, *DSP Based N-Motor Speed Control of Brushless DC Motors Using External FPGA Design*, IEEE, 2006

[28] Trinity Fire Fighting Robot Contest Site, http://www.trincoll.edu/events/robot

[29] Instituto Politécnico da Guarda, Fire Fighting Robot Contest Site, http://robobombeiro.ipg.pt

[30] X. Ji and Y. Lv, "Field weakening control of PMSM ued in a electric power steering system," in Electric Information and Control Engineering (ICEICE), 2011 International Conference on, 2011, pp. 2194-2199

[31]http://www.deetc.isel.ipl.pt/jetc05/CCTE02/papers/finais/fortes/22.pdf - 3 Fev 2013

[32] M. Zeraoulia, et al. "Electric motor drive selection issues for HEV propulsion systems: A comparative study," 2005 IEE Vehicle Power and Propulsion Conference (VPPC), pp. 280-287 - 837, 2005.

[33] J. B. D. d. O. Junior, "Avaliação comparativa de observadores em modo deslizante para acionamentos de máquinas síncronas de ímanes permanentes," RIO DE JANEIRO, 2007

[34] Microship, "Sensorless Field Oriented Control of PMSM Motors."

[35] S. A. M. Granadeiro, "Controlo de Motor Assíncrono Aplicado a Veículos Eléctricos," Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2009.

[36] Raul Rojas, "Omnidirectional Control", Freie Universität Berlim, 2005

[37] Modern-Electric-Hybrid-Electric-and-Fuel-Cell-Vehicles

[38] Mehrdad Ehsani, Yimi Gao, Sebastien E. Gay, Ali Emadi – Modern Electric, Hybrid Electric, and Fuel Cell Vehicles, Fundamentals, Theory, and Desgin

[39] Simple Analysis for Brushless DC Motors – Case Study: Razor Scooter Wheel Motor

[40] Richard C. Dorf – The Industrial Electronics Handbook Power Electronics and Motor Drives

[41]Jussi Puranen, Induction Motor Versus Permanent Magnet Synchronous Motor In Motion Control Applications: A Comparative Study

[42] AN857 – Microchip – Brushless DC Motor Control Made Easy

[43] http://wiki.freepascal.org/ - 3 Fev 2013

[44] http://www.trenz-electronic.de/products/fpga-boards/oho-elektronik.html - 3 Fev 2013

[45] http://higheredbcs.wiley.com/legacy/college/seborg/0471000779/dig_control/ch26.pdf - 3 Fev 2013

[46] http://blog.opticontrols.com/archives/383 - 3 Fev 2013

[47] http://www.mstarlabs.com/control/znrule.html - 3 Fev 2013

[48] K. J. Aström, T. Hägglund, Revisiting the Ziegler-Nichols step response method for PID control

[49] http://www.beyondlogic.org/usbnutshell/usb1.shtml - 3 Fev 2013

[50]A. Carvalhosa, P. Machado, A. Sousa, and J. C. Alves, "Soft core robot with joint wheel motion controller," in Industrial Electronics, 2009. IECON '09. 35th Annual Conference of IEEE, 2009, pp. 3168-3173.