UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

JEHOVAH TAVARES COELHO NETO

CONTROLE DE MOTOR CC USANDO O SOC-FPGA CYCLONE V

FORTALEZA
2019

JEHOVAH TAVARES COELHO NETO

CONTROLE DE MOTOR CC USANDO O SOC-FPGA CYCLONE V

Monografia apresentada ao Departamento de Engenharia Elétrica da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Bacharel em Engenharia Elétrica.

FORTALEZA

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

T23c Tavares Coelho Neto, Jehovah.
Controle de motor CC usando o SoC-FPGA Cyclone V / Jehovah Tavares Coelho Neto. – 2019.
41 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia Elétrica, Fortaleza, 2019.
Orientação: Prof. Dr. Bismark Claure Torrico.

1. SoC-FPGA. 2. Cyclone V. 3. Controlador PI. 4. Sintonia IMC. I. Título.

CDD 621.3

JEHOVAH TAVARES COELHO NETO

CONTROLE DE MOTOR CC USANDO O SOC-FPGA CYCLONE V

Monografia apresentada ao Departamento de Engenharia Elétrica da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Bacharel em Engenharia Elétrica.

Aprovada em: ___/___/_____.

BANCA EXAMINADORA

_____
Prof. Dr. Bismark Claure Torrico (Orientador)
Universidade Federal do Ceará (UFC)

_____
Prof. Dr. Fabrício Gonzalez Nogueira
Universidade Federal do Ceará (UFC)

_____
MSc. René Descartes Olímpio Pereira
Universidade Federal do Ceará (UFC)

A Deus.

Aos meus pais, Fernando e Francisca.

Aos meus irmãos, Nando (*in memorian*) e Pedro.

AGRADECIMENTOS

À minha família, em especial, ao meus pais e irmãos, pelo suporte e apoio em todos os momentos difíceis de minha vida.

Aos Professores Dr. Bismark Claure Torrico e Dr. Fabrício Gonzalez Nogueira por terem aceitado esse trabalho de orientação.

Ao participante da banca examinadora René Descartes Olímpio Pereira pelo tempo e pelas valiosas colaborações e sugestões.

Aos meus amigos e colegas do curso de Engenharia Elétrica, pelo convívio, pelas brincadeiras e por terem tornado esse caminho difícil um processo mais prazeroso.

“O que sabemos é uma gota; o que ignoramos é um oceano.”

Isaac Newton

RESUMO

Este trabalho apresenta o projeto de um controlador de velocidade PI para uma planta formada por um SoC-FPGA Cyclone V, da Intel, e um motor de corrente contínua acoplado a um encoder de quadratura. O controlador desenvolvido foi um controlador PI de velocidade baseado em um método de sintonia IMC de Skogestad. No SoC-FPGA foi desenvolvida uma aplicação mista entre o HPS e o FPGA presentes no chip. Na parte do FPGA foram desenvolvidos blocos em programação VHDL responsável pela leitura do encoder e geração de um PWM. Na parte do HPS, foi desenvolvido um código fonte em linguagem C responsável pelo loop do controlador.

Palavras-chave: SoC-FPGA. Cyclone V. Controlador PI. Sintonia IMC

ABSTRACT

This paper presents a project a PI speed controller for a plant formed by a SoC-FPGA Cyclone V, by Intel, and a DC motor linked to a quadrature encoder. The controller proposed was a PI speed controller based on an IMC tuning method developed by Skogestad. It was created a mixed application between the HPS and the FPGA inside the SoC-FPGA. It was developed VHDL blocks on Quartus software that they are responsible for receiving encoder data and to generate a PWM. It was also generated a Language C application responsible for the controller loop.

Keywords: SoC-FPGA. Cyclone V. PI controller. IMC tuning.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CI	Circuito Integrado
CPLD	*Complex Programmable Logic Device*
DSP	*Digital Signal Processor*
EPROM	*Erasable Programmable Read-Only Memory*
FPGA	*Field Programmable Gate Array*
GPIO	*General Purpose Input/Output*
HPS	*Hard Processor System*
IP	*Intellectual Property*
LB	*Logic Block*
LUT	*Look-Up Table*
PAL	*Programmable Array Logic*
PLA	*Programmable Logic Array*
PLD	*Programmable Logic Device*
PROM	*Programmable Read-Only Memory*
PWM	*Pulse-Width Modulation (Modulação por Largura de Pulso)*
ROM	*Read-Only Memory*
SoC	*System-on-a-Chip*
SSH	*Secure Shell* (Terminal Seguro)

LISTA DE SÍMBOLOS

V_{RPM}	Velocidade angular do motor em rotações por minuto
$N^o\ Pulsos$	Número de bordas de subida e de descida dos canais A e B em um período de amostragem de 0,01s
u(t)	Sinal de controle
e(t)	Erro do processo
K_P	Ganho Proporcional
T_I	Tempo Integral
k	Ganho da planta
Θ	Tempo de atraso da planta
τ_1	Constante de tempo do processo
τ_C	Parâmetro de ajuste de Skogestad

SUMÁRIO

1 INTRODUÇÃO

Sistemas de controle modernos precisam oferecer diferentes características para suprir as carências de um mercado tão rápido e de tanto impacto como o da tecnologia. A teoria de controle automático está cada vez mais integrada a área da computação. Novos algoritmos e recursos computacionais têm sido utilizados para realizar os mais diversos tipos de controle (MOROMISATO *et al*., 2007). Nesse contexto de evolução dos sistemas digitais, surgiram os SoC-FPGA's. Reunindo características dos mais modernos FPGA's e de processadores ARM de última geração num mesmo chip de silício, a família Cyclone V da Intel se tornou o principal elemento deste trabalho por representar uma aposta de sua utilidade em projetos mais complexos futuramente.

Como forma de medir o desempenho do chip da família Cyclone V, foi montada uma planta formada por um kit de desenvolvimento DE1-SoC, que possui um SoC-FPGA Cyclone V, e um motor de corrente contínua para a aplicação de um controlador em malha fechada. O controlador escolhido para o ensaio foi um PI, e, ainda, foi utilizado um método de sintonia para obtenção dos parâmetros do controlador. O tipo de planta e o controlador citados foram escolhidos por serem bastante comuns na indústria. De acordo com (ASTROM; HAGGLUND, 2001), O controle Proporcional-Integral-Derivativo (PID) é ainda hoje predominante no meio industrial. Mais de 90% de todas as malhas existentes são do tipo PI / PID atingindo uma larga faixa de aplicações: controle de processos, indústria automobilística, controladores de voo, pilotos automáticos, instrumentação, entre outros.

A divisão deste trabalho se inicia no capítulo 2 onde são fornecidas informações históricas que vão dos primeiros dispositivos lógicos criados até o dispositivo principal deste trabalho, o SoC-FPGA. O capítulo 3 trata especificamente da família Cyclone V da Intel. Neste, é visto as formas de se desenvolver aplicações para esta família além de características do kit de desenvolvimento DE1-SoC. O capítulo 4 apresenta os detalhes de conexão dos equipamentos e a forma de divisão do projeto entre hardware e software. Ainda, como a execução de cada passo foi feita. O capítulo 5 descreve, passo a passo, como foi executado o projeto do controlador PI desenvolvido e os resultados experimentais obtidos. Por fim, o capítulo 6 apresenta as conclusões deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Será mostrado nesse capítulo um breve resumo da história dos componentes eletrônicos que antecederam o principal elemento deste trabalho: o SoC-FPGA. Nesse resumo, será visto alguns dos mais importantes nomes da eletrônica e um pouco do funcionamento de suas criações. Além disso, será apresentado o conceito de SoC-FPGA e alguns dos principais fabricantes da atualidade.

2.1 Dispositivos Lógicos Programáveis Simples

A história dos primeiros dispositivos lógicos programáveis se inicia em 1956, antes mesmo de tais dispositivos executarem funções lógicas básicas, com a invenção das primeiras memórias PROM. Enquanto o conteúdo das memórias ROM, comuns na época, eram definidos já na sua fabricação, as primeiras memórias programáveis, uma revolução para a época, eram fabricadas sem nenhum conteúdo e sua memória era depois programada pelo usuário final. Dessa forma, ao aplicar tensões específicas nos fusíveis do circuito, um circuito final e definitivo correspondia à programação feita pelo usuário. Sua invenção é atribuída a Wen Tsing Chow a pedido da Força Aérea dos Estados Unidos e sua comercialização iniciou apenas em 1969.

Em 1971 aconteceu outra revolução no mundo da eletrônica com o surgimento das primeiras memórias PROM reprogramáveis: as memórias EPROM. Sua invenção é atribuída ao engenheiro eletricista Dov Frohman, que desenvolveu as primeiras peças da memória EPROM 1701 de 256 bytes nos laboratórios da Intel. A reprogramação desse dispositivo funcionava através de uma pequena janela de quartzo na parte superior do encapsulamento. Ao expor essa janela à luz ultravioleta, o padrão anteriormente gravado na peça era completamente apagado, deixando a memória novamente limpa. O marco representado por esta invenção foi tão grande que Gordon Moore, um dos fundadores da Intel e pai da famosa Lei de Moore, declarou numa revista anos depois, “a memória EPROM é provavelmente tão importante para o desenvolvimento da indústria de microcomputadores quanto para a própria indústria de microprocessadores” (INTEL, 1984, p. 22).

Com a introdução das novas memórias não-voláteis no mercado de semicondutores, o próximo passo foi a junção, num mesmo pedaço de silício, de uma memória EPROM e um circuito capaz de resolver equações lógicas simples: que ficou conhecido como PLA. Tendo como vantagem a possibilidade de substituir vários CI's discretos por um único CI, o PLA foi criado para tornar as fases de criação e teste de novos circuitos mais rápidas. Seu funcionamento se baseia em dois arranjos interligados de portas lógicas AND e portas lógicas OR. Através da programação das entradas e saídas, funções lógicas simplificadas na forma de soma de produtos podiam ser obtidas. Apesar disso, devido ao alto custo de produção e pelas dificuldades de usabilidade declarada pelos usuários, o PLA foi rapidamente substituído no mercado pelo PAL.

O novo dispositivo lógico veio para superar transtornos frequentes de seu antecessor como alto custo de produção e problemas ocasionados por um alto atraso de propagação do circuito. Os circuitos PAL foram criados em 1978 e, apesar também serem compostos por arranjos de portas OR e AND, os novos sistemas possuíam apenas o arranjo de portas AND com possibilidade de serem programados. Dessa forma, os novos circuitos são menos flexíveis que seus antecessores, porém mais confiáveis.

2.2 Dispositivos Lógicos Programáveis Complexos

O surgimento dos Dispositivos Lógicos Programáveis Complexos se deu quando múltiplos circuitos PAL foram interligados, através de conexões programáveis, em um único circuito integrado. De acordo com Brown e Rose (1996, p. 44), a empresa Altera, que se tornou parte da empresa Intel em 2015, foi pioneira na fabricação dos dispositivos CPLDs através dos chips da família Classic EPLD e, em seguida, da série Max 5000, 7000 e 9000. Mesmo com a criação de dispositivos mais modernos como o FPGA, que será discutido posteriormente, os Dispositivos Lógicos Programáveis Complexos continuam evoluindo até os dias de hoje de forma paralela aos FPGA. Entre as razões está o fato dos dispositivos CPLD serem mais baratos que os FPGAs de menor capacidade, além do fato deles não necessitarem de uma memória flash para executarem suas funções, ou seja, após a programação do usuário, eles estão pronto para uso.

2.3 Field Programmable Gate Arrays (FPGA)

Dentre os dispositivos mostrados até agora, o FPGA apresenta a maior densidade de portas lógicas de todos. Uma estrutura genérica de um FPGA simples consiste pelo menos de um conjunto Blocos Lógicos (LBs), uma rede de interconexões, além de blocos configuráveis de entrada e saída. Aproveitando do seu alto nível de densidade de componentes, "os dispositivos FPGA mais recentes já incluem blocos de memória, blocos de DSP, blocos gerenciadores de *clock*, além de blocos de comunicação, que suportam protocolos, como USB, Ethernet, CAN, PCI, SP" (MONMASSON, 2011, p. 3). Os LBs são responsáveis principalmente pelas operações combinacionais e sequenciais definidas pelo usuário. Para executar as funções combinacionais são considerados Lookup Tables (LUTs), e no segundo caso, são também utilizados *Flip-Flops* do tipo D. A rede de interconexões do FPGA também é programada pelo usuário e pode se conectar a quantos LBs forem necessários. A invenção deste dispositivo é atribuída a Ross Freeman e Bernard Vonderschmitt. Eles foram os cofundadores da empresa Xilinx e desenvolveram o primeiro modelo comercial em 1985, o XC2064.

2.4 SoC-FPGA

O SoC-FPGA representa uma solução ainda mais completa em termos de flexibilidade no design e capacidade "técnica" pois conjuga a versatilidade de um processador sintetizável, juntamente com um processador não-sintetizável que possui uma frequência de *clock* bastante superior aquele. Além disso, ele dispõe de diversos periféricos que tornam a construção de projetos realmente complexos mais simples e rápida. Os principais fabricantes da atualidade são a Intel, Xilinx e a Microchip. Eles fornecem diversas famílias de SoC-FPGA que variam desde os periféricos presentes no chip e blocos lógicos no FPGA até a inclusão de um processador ARM Quad-Core de 64 bits de extrema capacidade, como é o caso da família Stratix 10 da Intel.

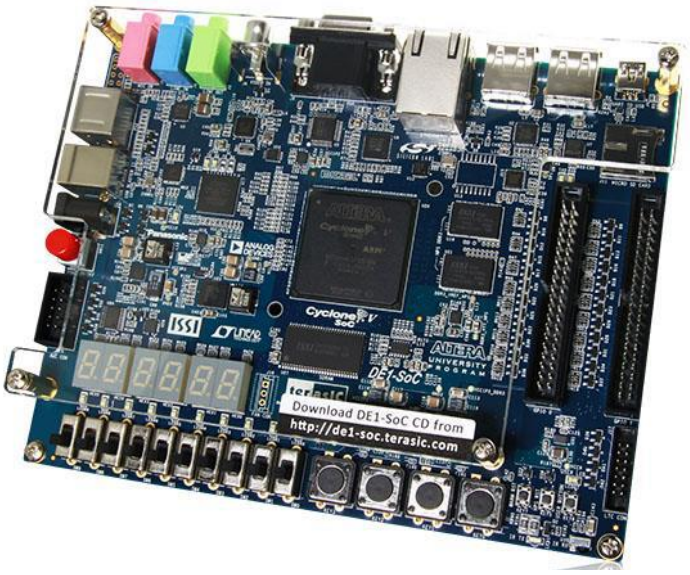
3 FAMÍLIA INTEL CYCLONE V FPGA

Este capítulo tem por objetivo apresentar o funcionamento geral da família de SoC-FPGAs: Cyclone V da Intel, mais especificamente, dos modelos utilizados na placa Terasic DE1-SoC. Serão mostradas inicialmente características gerais da placa, como: diagrama de blocos do SoC e periféricos presentes na mesma. Em seguida, serão apresentadas todas as ferramentas necessárias para o funcionamento conjunto do FPGA com o ARM Cortex-A9 presente neste SoC. A compreensão do funcionamento do dispositivo é de fundamental importância para o entendimento do projeto desenvolvido, que será apresentado no capítulo seguinte.

3.1 Placa de desenvolvimento Terasic DE1-SoC

Com o objetivo de estudar as possibilidades da fusão de arquiteturas de um processador ARM e um FPGA em um mesmo pedaço de silício, escolheu-se uma placa de desenvolvimento contendo o maior número de periféricos possível e, ao mesmo tempo, que já contenha os conectores necessários para a interface da placa com os sensores e atuadores do sistema proposto. Além disso, a placa de desenvolvimento Terasic DE1-SoC foi escolhida por possuir um baixo custo quando comparada com os demais kits de desenvolvimento pesquisados. A Figura 3.1, a seguir, mostra uma foto da placa.

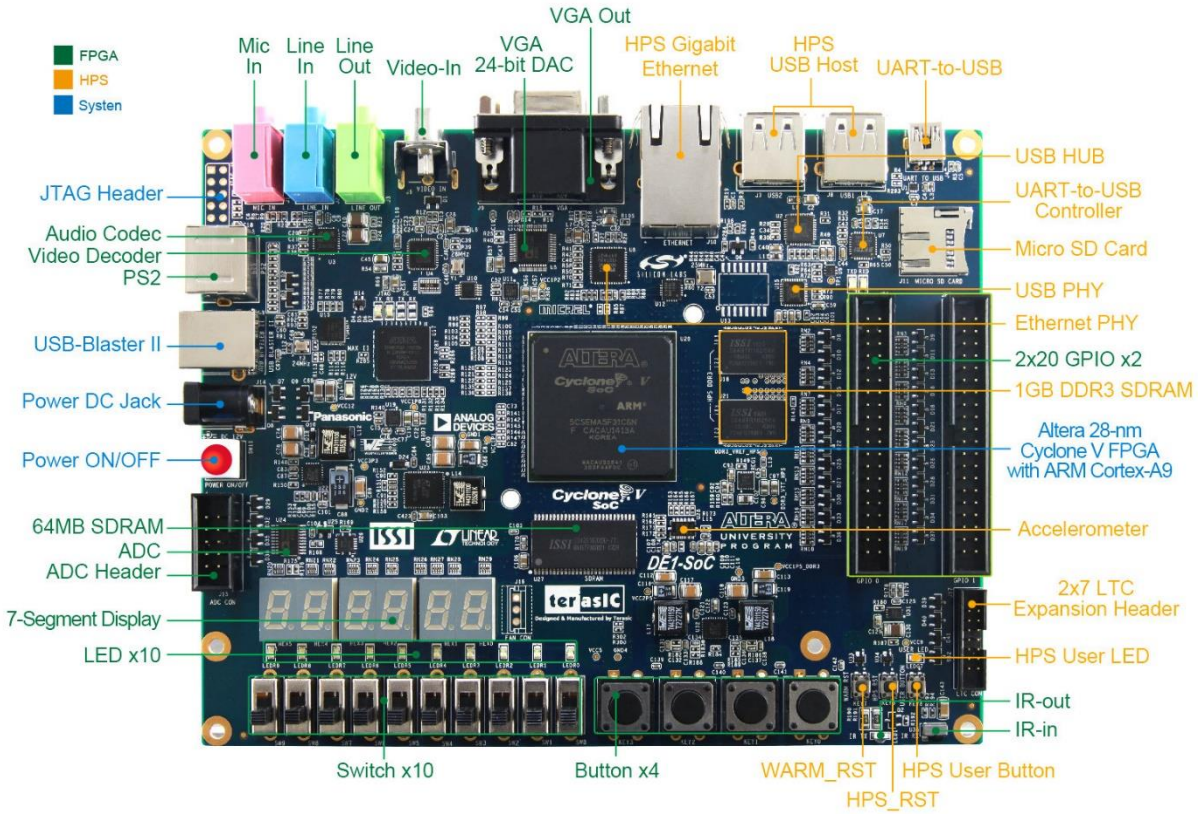
Figura 3.1 – Placa de desenvolvimento DE1-SoC



Fonte: Site da Terasic Technologies Incorporated (www.terasic.com)

O chip da família Cyclone V contido na placa, modelo 5CSEMA5F31C6, possui diversas funcionalidades e periféricos. Algumas delas formam o que a Intel chama de HPS, ou *Hard processor system*, já outras funções, fazem parte do próprio FPGA. O HPS é formado pelo processador Dual-core ARM Cortex-A9 e seus periféricos, como uma controladora de memória RAM e suporte a protocolos de comunicação UART, SPI e I2C. Muitos dos periféricos, tanto os do HPS como os do FPGA, já possuem IP cores gratuitos, ou seja, blocos lógicos pré-programados que servem de ponte entre eles e o sistema principal. A Figura 3.2 mostra a distribuição dos periféricos presentes na placa DE1SoC. Eles estão divididos entre verde e amarelo, onde verde são os periféricos ligados ao FPGA, e amarelo, os periféricos ligados ao HPS.

Figura 3.2 – DE1-SoC e seus periféricos



Fonte: Site da Terasic Technologies Incorporated (www.terasic.com)

3.2 Funcionamento do SoC Cyclone V

O funcionamento do dispositivo está relacionado com a forma de utilização do usuário, que pode ser dividida de três formas distintas: utilização individual do FPGA, utilização individual do HPS ou a combinação de ambas as partes do chip.

Utilizando somente o FPGA, o usuário poderá fazer suas aplicações normalmente em linguagens de descrição de Hardware comuns como Verilog e o VHDL, além de poder usufruir dos IP cores disponíveis no software QSYS. Este software faz a integração de subsistemas desenvolvidos pelo usuário de forma bastante eficiente, tornando mais fácil o uso dos periféricos presentes neste kit e em outros. Numa aplicação que utilize ambos, o usuário pode escolher entre duas formas de trabalho: fazer uma aplicação *Bare-Metal* ou fazer uma Aplicação em Linux. Seu detalhamento será feito a seguir. E finalmente, utilizando somente o HPS em sua aplicação, o usuário possui as mesmas opções e que na forma anterior, mas sem a utilização de subsistemas escritos em linguagem de descrição de hardware.

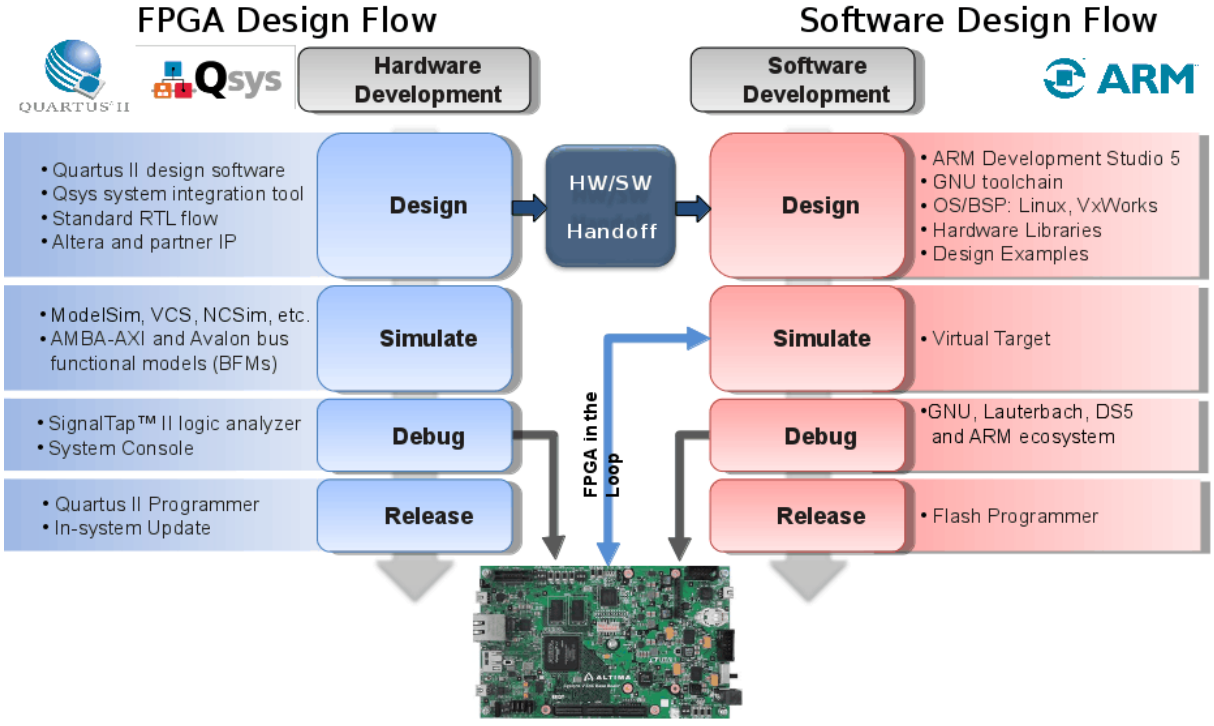
3.2.1 Aplicação Bare-metal

Este é um tipo de aplicação de baixo nível, onde programador tem acesso real a memória física do processador ARM, utilizando-o de forma semelhante a um microcontrolador comum. Esta forma de trabalho possui diversas vantagens como o acesso total do usuário ao software gerado e a possibilidade de gerar um código compilado mais enxuto e preciso. Além disso, como no caso do Cyclone V, o fato de não haver um sistema operacional que rode acima do código desenvolvido, permite uma execução mais rápida e segura da aplicação. Uma grande desvantagem nessa forma de trabalho é o tempo consideravelmente maior para o desenvolvimento de uma aplicação simples, visto que o programador é responsável pela elaboração de um sistema desde o início, incluindo a criação dos drivers dos periféricos. No caso dos dispositivos SoC-FPGAs da Intel, como o Cyclone V, uma outra desvantagem é necessidade de se obter uma licença paga para a utilização do software de desenvolvimento ARM DS-5. O ARM DS-5 é um ambiente de criação de aplicações bare-metal ou em Linux para processadores ARM. Ele possui uma IDE baseado no Eclipse, cobrindo todos os estágios de criação de uma aplicação, incluindo a depuração.

3.2.2 Aplicação em Linux

Este é um tipo de aplicação de alto nível, a qual se faz necessário rodar a aplicação dentro de um sistema operacional previamente configurado. Para os dispositivos da Intel, são disponibilizadas distribuições Linux, como a versão Ubuntu utilizada neste trabalho, que já são previamente configuradas para rodar em um SoC-FPGA. Conforme foi dito anteriormente, o processo de criação de aplicações em Linux é mais rápido, porém menos seguro. Isso ocorre porque a distribuição Linux, na qual a aplicação a ser desenvolvida pelo usuário rodará, só permite o acesso indireto do programador aos periféricos do sistema através do mapeamento da memória física. Neste mapeamento, o usuário somente tem acesso a partes da memória por vez, onde periféricos como: Controlador de I2C, GPIO, Controladora de UART, possuem endereços fixos e seu acesso deve ser encerrado ao final do programa, diferentemente da aplicação *Bare-Metal*, onde o acesso pode se dar a qualquer momento e alterando qualquer parte da memória que se queira. A Figura 3.3 abaixo mostra o fluxo de trabalho de uma aplicação em Linux para um SoC-FPGA da Intel.

Figura 3.3 – Fluxo de trabalho de uma Aplicação em Linux



Fonte: Site da Terasic Technologies Incorporated (www.terasic.com)

Como pode ser visto na Figura 3.3, a criação de uma aplicação em Linux para um SoC-FPGA se divide em dois processos paralelos:

- Desenvolvimento de Hardware;
- Desenvolvimento de Software.

No desenvolvimento de hardware, tem-se a criação de todos os blocos em linguagem de hardware que servirão de apoio para o design final. No caso deste trabalho, existem blocos de leitura de sensores, recebimento e envio de dados, e criação de um PWM. Além desses, e mais importante, existe a criação e configuração do modo de funcionamento do HPS, utilizando o software QSYS. Nela, se definem os periféricos a serem utilizados, seus respectivos pinos, a forma de comunicação em o HPS e o FPGA, etc. Após esta etapa, são gerados os *Handoff Files*, que são os arquivos que transcrevem para o HPS o que foi feito no FPGA, permitindo o avanço para o processo seguinte.

No desenvolvimento de software, a criação de uma nova aplicação se dá a partir dos *Handoff Files* e de bibliotecas disponíveis pela Intel compostas de funções que facilitam a programação do HPS. Para um acesso, por exemplo, a um dos módulos GPIO do Cyclone V, alguns passos devem ser seguidos para a utilização do mapeamento de memória, conforme Tabela 3.1.

Tabela 3.1 – Funções usadas no Mapeamento da Memória Física

Função	**Ação**
open	Abrir o driver do dispositivo que terá a memória mapeada
mmap	Mapear a memória física para o espaço do usuário
alt_read_word	Ler o valor de um determinado registrador
alt_write_word	Escrever um valor em um determinado registrador
munmap	Limpar o mapeamento de memória
close	Encerrar o acesso ao driver do dispositivo

Fonte: elaborada pelo autor.

Para o acesso a qualquer um dos módulos do HPS, deve-se primeiramente observar a lista de endereços do HPS no datasheet do SoC. No caso do GPIO, pode se observar na Figura 3.4 que o SoC Cyclone V possui três GPIOs, e onde estão localizados seus endereços. A partir do endereço no datasheet e do *offset* obtido nos *Handoff Files* do QSYS, basta a criação de um programa simples em Linguagem C que utilize corretamente a função mmap, descrita na Tabela 3.1, para o acesso do usuário aos módulos do HPS.

Figura 3.4 – Mapa de Endereços do HPS

HPS

Identifier: HPS
Access: R/W
Description: Address map for the HHP HPS system-domain

Title	Identifier	Offset
Reserved		0x0
QSPI Flash Controller Module Register	QSPIREGS	0xFF705000
		0xFF705100
Manager Module	FPG	
ACP ID Mapper Registers	ACPIDMAP	
GPIO Module	GPIO0	0xFF708000
Reserved		0xFF708080
GPIO Module	GPIO1	0xFF709000
Reserved		0xFF709080
GPIO Module	GPIO2	0xFF70A000
Reserved		0xFF70A080
L3	EGS	0xFF800000
		0xFF880000
AND Controller Module Data (AXI Slave)	NAND	
EMAC Module	EMAC1	0xFF702000

Fonte: Site da Terasic Technologies Incorporated (www.terasic.com)

Exemplos mais completos de acesso aos periféricos do chip utilizando o mapeamento de memória são fornecidos pela Intel, fabricante do SoC-FPGA, através de sua comunidade de discussão em Rocketboards e pela Terasic, fabricante da placa DE1-SoC.

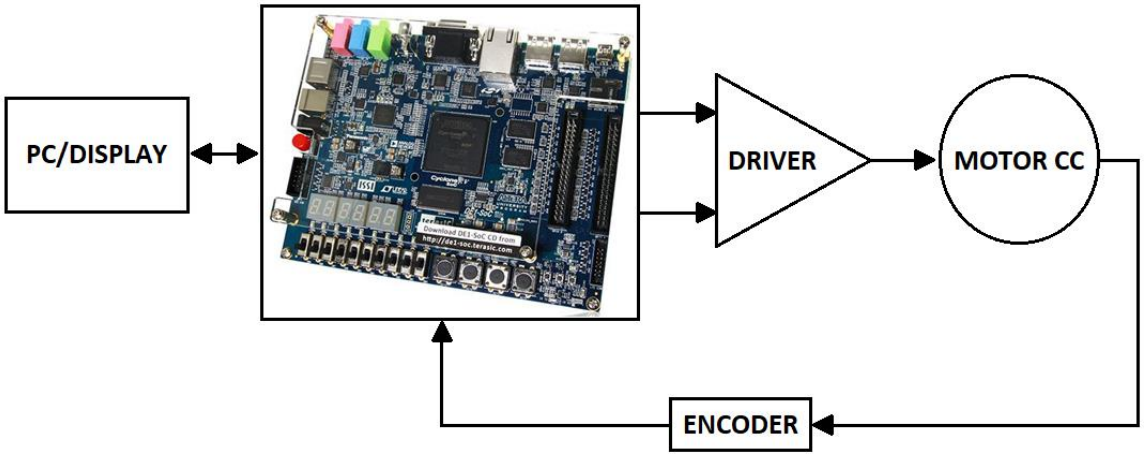
4 IMPLEMENTAÇÃO DO SISTEMA

Este capítulo apresenta todas etapas de construção do protótipo, desde o diagrama físico, até detalhes do algoritmo de controle do motor. Primeiramente será visto como os componentes físicos do sistema estão ligados. Depois disso será mostrado as duas partes na qual o projeto se divide: Implementação de Hardware e Implementação de Software.

4.1 Representação do sistema físico

A Figura 4.1 mostra uma representação em diagrama de blocos do sistema proposto. Nela, tem-se a placa DE1-SoC como principal componente, sendo responsável pela leitura de velocidade do encoder, processamento e execução do algoritmo do controlador, além da comunicação com um laptop que mostra em tempo real os dados de velocidade do motor.

Figura 4.1 – Diagrama de Blocos do Sistema Proposto



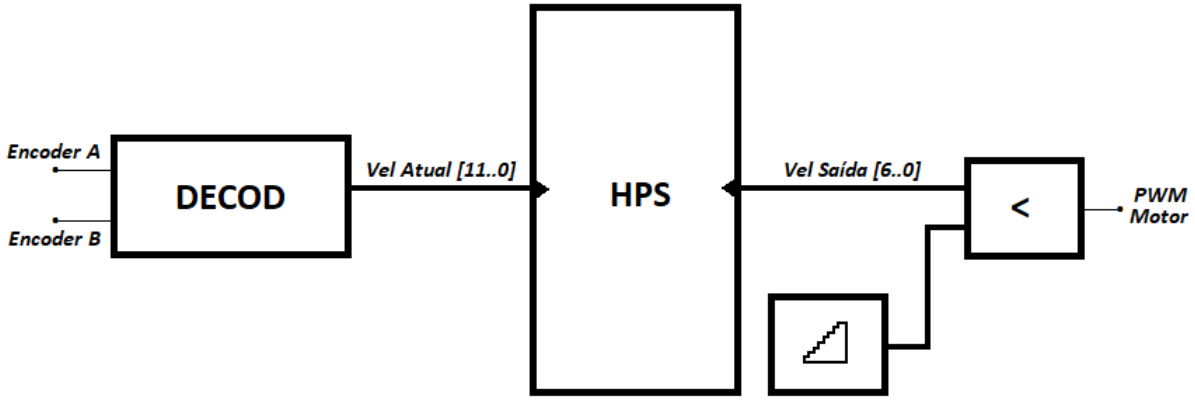
Fonte: elaborada pelo autor.

Seguindo o que foi apresentado anteriormente em *3.2.2*, o processo de construção de um projeto no SoC-FPGA Cyclone V pode ser dividido em duas partes: criação dos blocos em linguagem de descrição de hardware ligados ao FPGA, e criação de um algoritmo em C que seja executado no sistema operacional Linux instalado no HPS. Para este projeto, a primeira parte pode ser subdividida em: leitura de velocidade do encoder, configuração do HPS e geração do PWM. Já a segunda parte é subdividida na comunicação da placa com o PC e na implementação do algoritmo do controlador PI do motor. Seu detalhamento é feito a seguir.

4.2 Implementação de Hardware no SoC Cyclone V

A Figura 4.2 mostra de forma resumida o projeto de hardware desenvolvido no software Quartus. Nela, pode ser visto que o bloco denominado DECOD é responsável pela entrada de dois sinais digitais do encoder e geração de uma saída de 12 bits correspondente à velocidade atual do Motor CC. Ainda, que a configuração do HPS através do software QSYS deve passar pela criação de duas portas de dados: uma porta de entrada de 12 bits e uma porta de saída de 7 bits. E, por fim, que a geração de um sinal PWM aplicável ao motor necessita de dois blocos adicionais: um responsável pela criação de uma onda dente de serra e um bloco comparador.

Figura 4.2 – Resumo do Projeto de Hardware



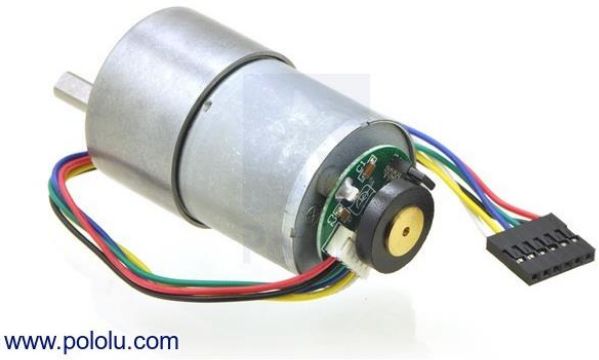
Fonte: elaborada pelo autor.

4.2.1 Decodificação do Encoder de Quadratura

Com o intuito de evitar problemas de escorregamento entre o encoder e o motor, optou-se por um sistema onde ambos já estejam acoplados, como mostra a Figura 4.3. Esse sistema é formado por um Motor CC de 12V acoplado a um conjunto de engrenagens de redução na proporção de 131,25:1 e soldado a um encoder de quadratura de 2 canais com uma resolução de 64 contagens por volta. Através da caixa de engrenagem, a resolução final do sistema se torna 8400 contagens por volta.

Para se chegar à resolução de 8400 contagens por revolução, se faz necessário contar o número de bordas de subida e descida de ambos os canais. Dessa forma, é possível se calcular a velocidade angular do motor fazendo a contagem do número de bordas de subida e descida em um determinado período de amostragem como será mostrado a seguir.

Figura 4.3 – Motor CC utilizado no Projeto



Fonte: Site da Pololu (www.pololu.com).

O funcionamento do encoder escolhido se baseia em um sensor Hall, que requer uma tensão de entrada Vcc de 3,5V à 20V (diferente da alimentação do motor), gerando em seus 2 canais de saída A e B ondas quadradas de frequência variável com defasagem de aproximadamente 90º, e de tensão 0 – Vcc. A Figura 4.4 mostra a captura de um osciloscópio onde uma tensão de 12V é aplicada no motor e uma tensão Vcc de 5V é aplicada ao encoder.

Figura 4.4 – Captura dos Canais A e B do Encoder

Fonte: elaborada pelo autor

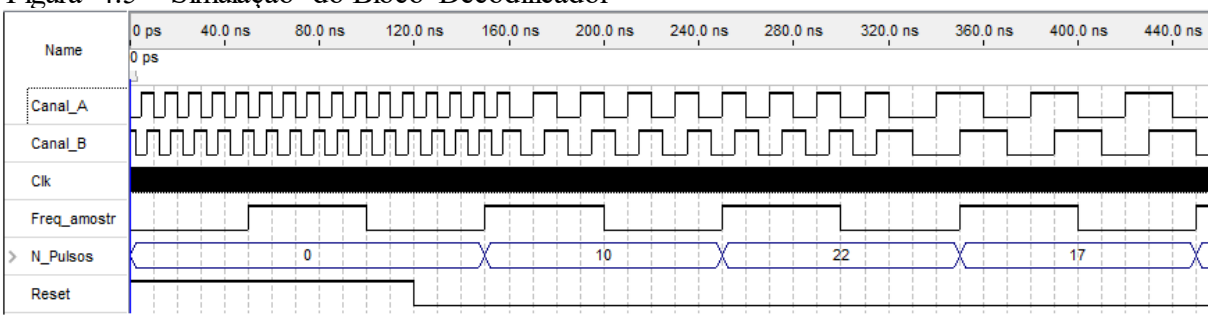
A partir do modo de funcionamento do encoder mostrado anteriormente, criou-se 2 blocos de código em VHDL capazes de interpretar os sinais vindos dos canais A e B, como mostra o fluxograma da Figura 4.6 de forma simplificada. Nele, é possível ver que o bloco decodificação recebe os 2 sinais de saída do encoder, fazendo a contagem das bordas de subida e descida de ambos e enviando tal informação para um segundo bloco velocidade. Este, entrando em funcionamento apenas na frequência de amostragem de 100Hz, carrega o atual valor da contagem e faz a subtração com o valor anterior armazenado. Dessa forma, com o número de bordas armazenados em um período de amostragem, calcula-se a velocidade angular do motor da seguinte forma:

$$V_{RPM} = \frac{\text{Nº pulsos X } 86}{120} \tag{4.1}$$

Na Equação 4.1: "Nº de pulsos" é a quantidade de bordas de subida ou de descida dos canais A e B em um período de amostragem de 0,01s (Frequência de amostragem de 100 Hz), 86 é a velocidade angular máxima do motor em rotações por minuto e 120 é o número máximo de pulsos que podem ser gerados em um período de amostragem. A equação acima, porém, é feita somente no algoritmo de controle no processador HPS, como mostra o fluxograma.

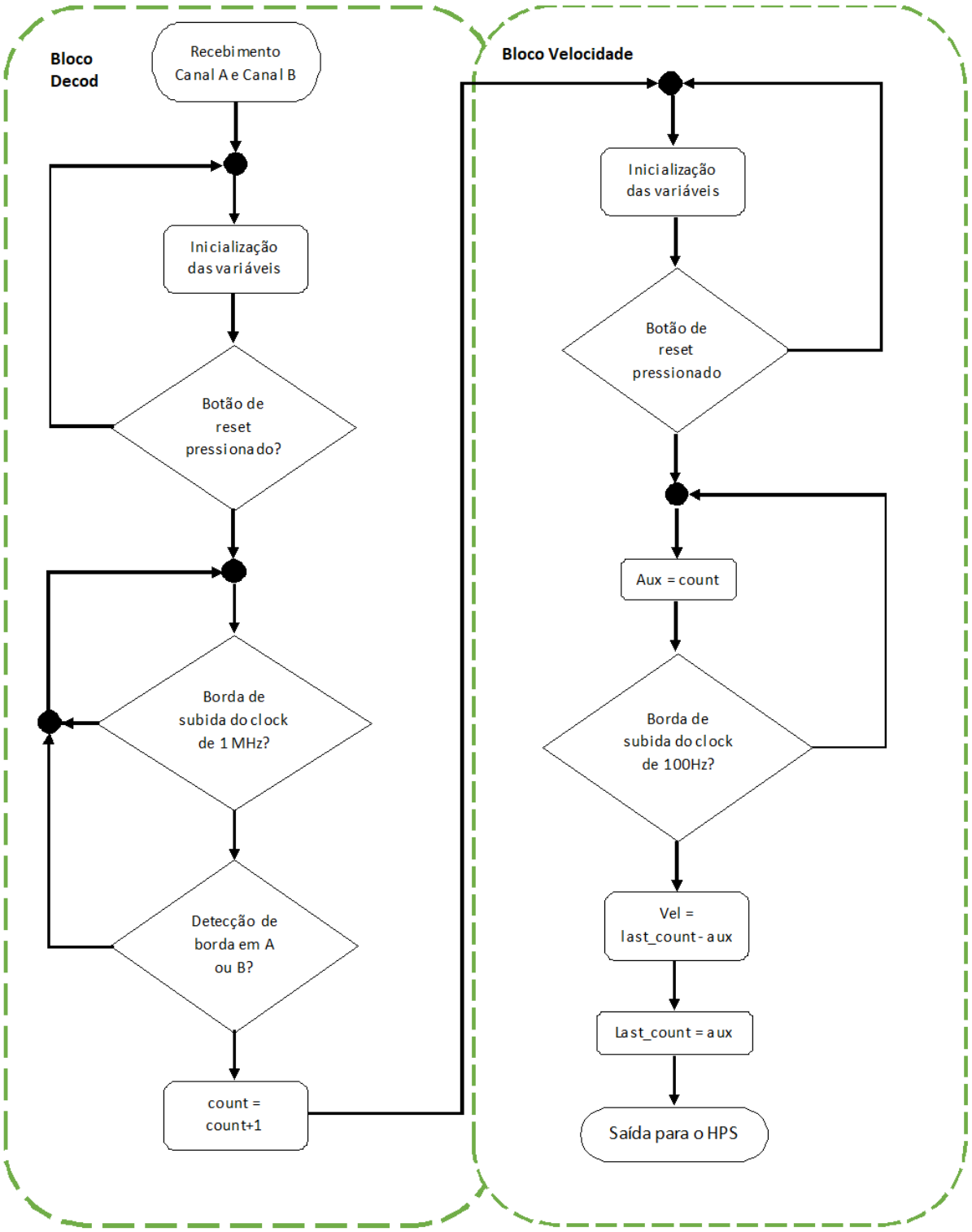
Para mostrar o funcionamento do bloco de decodificação, utilizou-se a ferramenta de simulação *Simulation Waveform Editor*. Atribuindo sinais aleatórios aos Canais A e B, simulando a leitura do encoder, e inserindo uma frequência de amostragem suficientemente razoável que permita a contagem de pulsos no gráfico, pode-se observar na Figura 4.5 que a cada borda de subida do sinal "Freq_amostr", o valor de saída em "N_Pulsos" corresponde a soma do número de bordas de subida e de descida dos sinais: "Canal_A" e "Canal_B".

Figura 4.5 – Simulação do Bloco Decodificador



Fonte: elaborada pelo autor.

Figura 4.6 – Fluxograma do Bloco Decodificador



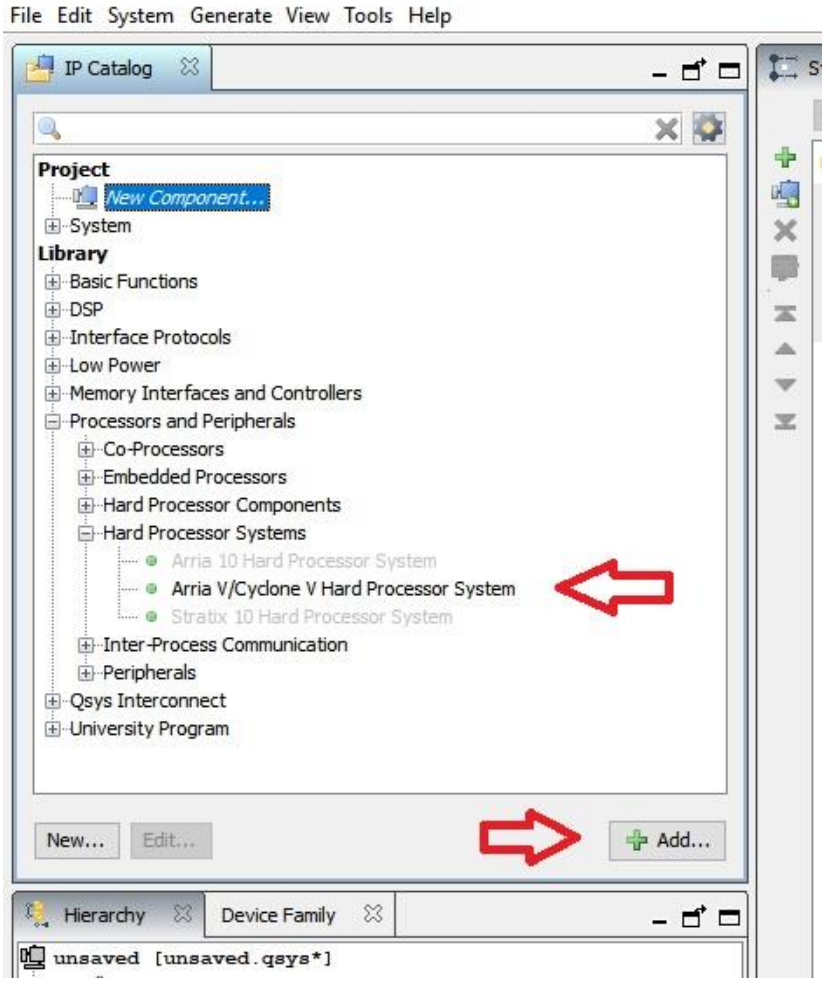
Fonte: elaborada pelo autor.

4.2.2 Configuração do HPS no software QSYS

A próxima etapa na Implementação de Hardware é a criação e configuração do HPS no software QSYS. Sendo parte fundamental para o entendimento deste trabalho, será detalhado nesse capítulo o funcionamento e configuração do software QSYS.

A criação do HPS no software QSYS se inicia com a escolha do FPGA utilizado no projeto, neste caso, o kit escolhido DE1-SoC possui um Cyclone V. Para fazer a escolha, ao abrir o software QSYS, é possível ver na aba do canto superior esquerdo, um Catálogo de IP's, ou seja, blocos de código previamente desenvolvidos pela Altera, que podem ser utilizados em aplicações maiores. Ao selecionar a biblioteca *Processors and Peripherals*, sub-itens *Hard Processor Systems ->Arria V/Cyclone V Hard Processor System*, deve-se clicar em *Add* para adicioná-lo ao sistema, como mostra a Figura 4.7.

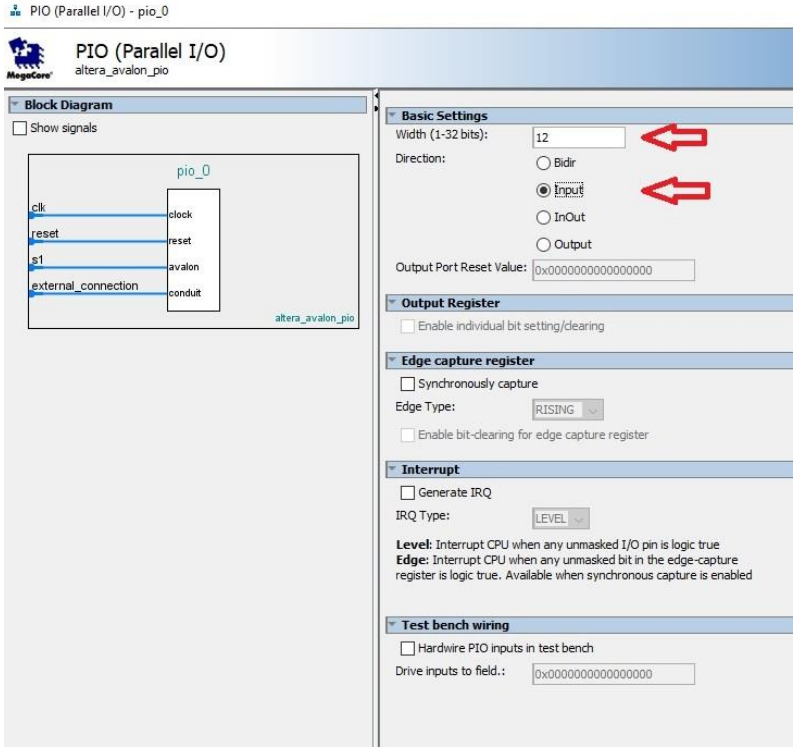
Figura 4.7 – Escolha do HPS no QSYS



Fonte: elaborada pelo autor.

Após a adição do HPS ao sistema, se faz necessário adicionar duas portas, sendo uma de entrada de dados, para o recebimento de 12 bits de dados de velocidade do encoder; e uma segunda porta de saída de dados de 7 bits, que representará um valor de velocidade de referência a ser alcançada pelo controlador desenvolvido. Para isso, deve-se selecionar a biblioteca *Peripherals*, subitem *PIO (Parallel I/O)*, e clicar em *Add*, adicionando uma porta ao sistema, como mostra a Figura 4.8.

Figura 4.8 – Adição das Portas de Dados



Fonte: elaborada pelo autor.

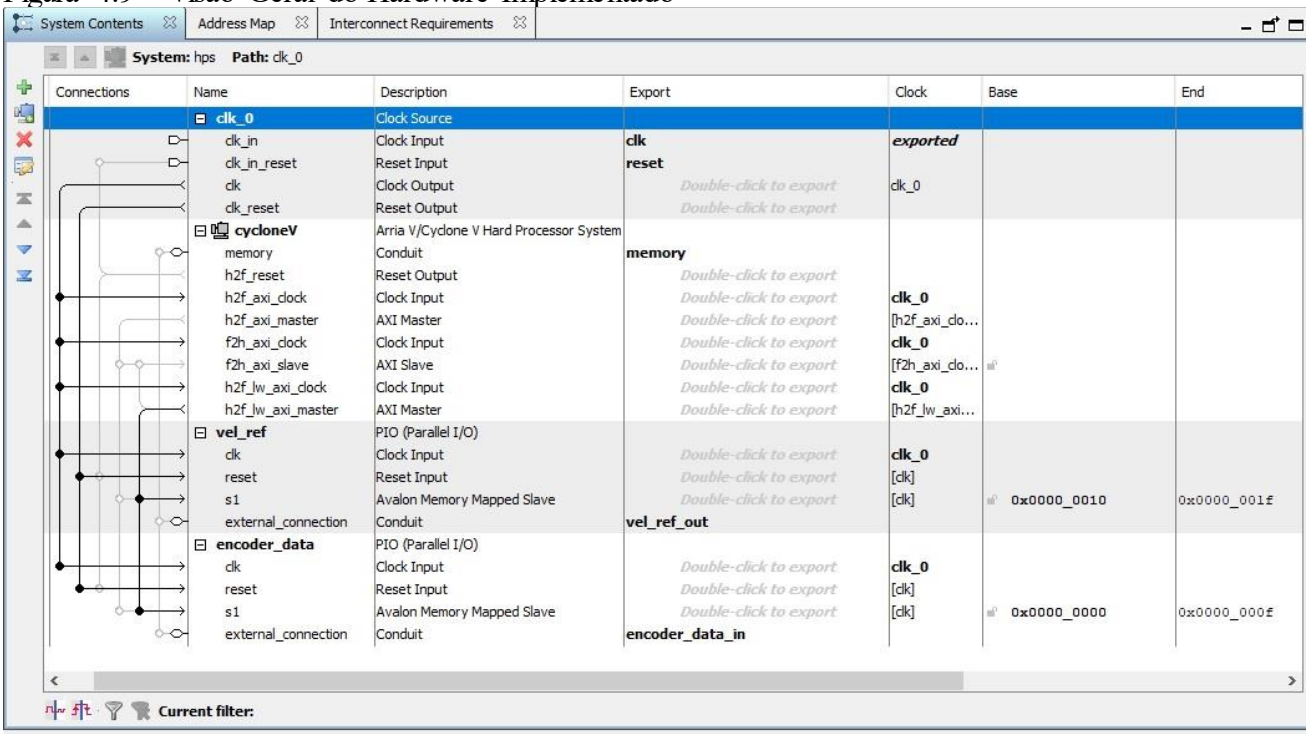
Na figura acima, é possível ver através das indicações que o tipo de direção dos dados escolhida foi entrada e que o tamanho da porta é de 12 bits. Para se adicionar a segunda porta, repete-se o procedimento anterior de escolha do item PIO, alterando apenas o tipo de direção dos dados para saída e o número de bits para 7, conforme dimensionado anteriormente.

Após a adição das portas de dados, o sistema final com as conexões já feitas é mostrado na Figura 4.9. Deve-se observar na aba *Name* que o sistema possui 4 principais núcleos: clk_0, cycloneV, vel_ref e encoder_data. O primeiro deles é gerado automaticamente pelo software e é responsável por gerar o clock para todos os demais blocos, tornando o sistema todo síncrono.

Na aba *Connections,* pode-se observar que a conexão entre o HPS e as portas de dados é feita entre um canal AXI Master, com o nome de h2f_lw_axi_master e os escravos como o nome de: *Avalon Memory Mapped Slave*. Cada porta funciona como um escravo na comunicação com o HPS, possuindo um endereço específico, como pode ser visto nas abas *Base* e *End* da Figura 4.9. Os endereços obtidos serão os mesmos a serem utilizados na Aplicação em Linux que conterá o controlador PI em Linguagem C.

Após a configuração do HPS e dos periféricos do sistema, o software QSYS fornece duas opções de escolha do que fazer com o sistema gerado: a primeira seria compilá-lo, transformando-o em um arquivo texto em Linguagem VHDL como entidade maior do projeto; a segunda seria transformá-lo num bloco a ser utilizado por uma entidade maior. Como mostra a Figura 4.2, o HPS é apenas mais um bloco do sistema, logo a segunda opção foi a utilizada neste projeto.

Figura 4.9 – Visão Geral do Hardware Implementado



Fonte: elaborada pelo autor.

4.2.3 Geração do PWM do Motor CC

A última etapa na Implementação de Hardware é a criação de blocos de código em VHDL que transformem um valor de velocidade do Motor CC obtido do Controlador PI, em um *dutycycle* de um PWM, transformando os valores digitais do controlador em um sinal analógico correspondente de 0V-5V.

Primeiramente, criou-se um bloco que faz uma contagem progressiva de 0 a 120, que é a quantidade máxima de pulsos que o encoder pode capturar em um período de amostragem. Um segundo bloco chamado comparador recebe a saída do bloco criado anteriormente e a saída do HPS criado em *4.2.2*, ou seja, o sinal de controle do Controlador PI. Ao utilizar da comparação de valores, o sinal obtido na saída do comparador possui um *dutycycle* proporcional ao valor do sinal do controlador. Esse sinal é posteriormente atribuído a um pino qualquer do kit através do menu Pin Planner do Quartus, tornando-o acessível para o driver do motor ou para uma eventual leitura com o osciloscópio.

4.3 Implementação de Software no SoC Cyclone V

Conforme foi dito em *3.2* existem duas formas de utilização dos processadores ARM no Cyclone V: a primeira através de uma Aplicação Bare-Metal de baixo nível, onde o usuário tem acesso imediato a cada registrador do processador, e a segunda, sendo esta a forma utilizada neste trabalho, através de uma Aplicação em Linux, onde uma distribuição do sistema operacional Linux roda no sistema e um programa que possui apenas o acesso indireto aos periféricos roda dentro do Linux.

Para este trabalho, a Aplicação em Linux é um programa escrito em Linguagem C que contém um loop de aplicação do Controlador PI do motor. Para usar de forma correta a função do mapeamento de memória do HPS, foram utilizados os exemplos disponíveis pela Terasic nos manuais da placa. Para facilitar a obtenção dos dados de saída do sistema, utilizou-se uma conexão SSH entre o PC e a placa DE1-SoC, sendo possível controlar remotamente, através do software ARM DS-5, a execução do programa Controlador PI. Com isso, sem utilizar nenhuma forma de armazenamento de dados na placa e usando apenas a função “printf” na aplicação, é possível acompanhar em tempo real os valores de velocidade do motor.

5 CONTROLE DE VELOCIDADE APLICADO A UM MOTOR DE CORRENTE CONTÍNUA

Este capítulo apresenta o projeto de um controlador PI de velocidade para o motor do sistema. Foi desenvolvido um controlador PI baseado na regra de sintonia pelo método IMC de Skogestad. Tendo em vista que o foco de trabalho é a demonstração de utilização de tecnologias mais atuais, como é o caso do SoC-FPGA, ao invés da busca pelo controlador mais eficiente, optou-se por um método de sintonia de controlador mais simples, apenas como uma forma de validação do sistema.

5.1 Introdução aos Controladores PI e PID

5.1.1 Controlador PI

O Controlador Proporcional-Integral (PI) é uma técnica de controle caracterizado pela atuação conjunta da ação Proporcional e da ação Integral sobre o erro. A forma com que esse controlador pode ser ajustado é definida por dois fatores conhecidos como K_P, ou ganho do controlador, e T_I, ou tempo integral. A Equação 5.1 demonstra o sinal de controle no tempo continuo e a função de transferência para esse tipo de controlador.

$$u(t) = K_P.e(t) + \frac{K_P}{T_I}\int_0^t e(t).d(t) \xrightarrow{L} \frac{U(s)}{E(s)} = K_P.\frac{(T_I.s+1)}{T_I s} \qquad (5.1)$$

Na equação acima, *u(t)* é a saída do controlador, K_P é o ganho proporcional e *e(t)* é o erro da variável do processo dado por $e(t) = y_R - y(t)$ onde y_R é a referência ou valor desejado e *y(t)* é a valor de saída do processo. O termo L representa a passagem da equação no domínio do tempo (t) para o domínio da frequência (s).

Segundo Ogata [7], o tempo integral ajusta a ação do controle integral, enquanto uma mudança na constante K_P afeta tanto a parte proporcional quanto a parte integral da ação de controle. Além disso, ele afirma que o inverso do tempo integral mede o número de vezes por minuto que a parte proporcional da ação de controle é duplicada. Medida essa que é chamada de taxa de restabelecimento. A principal função da parte integral da função de controle é garantir que a saída do processo atinja a referência em regime permanente [1].

5.1.2 Controlador PID

O controlador Proporcional-Integral-Derivativo (PID) é uma técnica de controle formada pela ação de três termos distintos, são estes: ação Proporcional, ação Integral e ação Derivativa. De forma complementar ao que foi dito sobre o Controlador PI, o propósito da ação derivativa é aumentar a estabilidade do sistema em malha fechada [1]. As Equações 5.2 e 5.3 demonstram, respectivamente, o sinal de controle no tempo continuo e a função de transferência para esse tipo de controlador.

$$u(t) = K_P.e(t) + \frac{K_P}{T_I}\int_0^t e(t).d(t) + K_P T_D \frac{de(t)}{dt} \quad \xrightarrow{L} \tag{5.2}$$

$$\frac{U(s)}{E(s)} = K_P.\left(1 + \frac{1}{T_I s} + T_D.s\right) \tag{5.3}$$

Existem diversas variações da forma de executar o algoritmo PID, como a forma apresentada na Equação 5.3. Cada diferença na forma com que os termos são ligados entre si altera substancialmente a performance do sistema [1]. Uma outra forma bastante utilizada na indústria é a forma PID-Série. A Equação 5.5 demonstra a função de transferência para esse tipo de controlador.

$$para\ D(s) \cong \frac{1}{1 + T_F.s} \tag{5.4}$$

$$\frac{U(s)}{E(s)} = K_P.\left(\frac{T_I.s + 1}{T_I.s}\right)\left(\frac{T_D.s + 1}{T_F.s + 1}\right) \tag{5.5}$$

É introduzido o filtro na ação derivativa, *D(s),* tornando o controlador possível de ser implementado em um equipamento físico (pneumático ou eletrônico analógico) [4]. O fator T_F costuma ser um valor pequeno, fazendo com que o numerador, ou seja, a ação derivativa, prepondere.

5.2 Métodos de Sintonia de Controladores

Existe atualmente um grande número de métodos de sintonia de controladores PID disponíveis na literatura. Eles se tornaram muito utilizados na indústria devido a maior rapidez de obtenção dos parâmetros, onde, em muitos casos, somente é necessário um rápido ensaio em malha aberta da planta para que os parâmetros de um controlador básico possam ser conseguidos e posteriormente otimizados. A seguir é apresentado o método desenvolvido por Sigurd Skogestad de sintonia de controladores.

5.2.1 Método de Sintonia de Skogestad

O método proposto por *Skogestad* [10] é uma técnica de sintonia baseado no método IMC conhecido por *Simple Internal Model Control* (SIMC). Nele, é possível determinar os valores de Ganho Proporcional (K_P), Tempo Integral (T_I) e Tempo Derivativo (T_D), com base apenas no modelo da planta e em parâmetros de ajuste. A Tabela 5.1 apresenta os valores de K_P e T_I para um modelo de primeira ordem com atraso, que foi o caso desenvolvido nesse trabalho.

Tabela 5.1 – Regra de sintonia do método SIMC

Modelo	**G(s)**	**K_P**	**T_I**	**T_D**
1º Ordem	$k\ \frac{e^{-\Theta s}}{(\tau_1 s+1)}$	$\frac{1}{k}\ \frac{\tau_1}{\tau_C+\Theta}$	$\text{mín}\{\tau_1, 4(\tau_C+\Theta)\}$	-

Fonte: elaborada pelo autor.

A linha da Tabela 5.1 corresponde a um controlador PI para um modelo integrador com atraso. Os valores K_P e T_I são totalmente determinados por k, Θ, τ_1, e τ_C, que são respectivamente: o ganho da planta, tempo de atraso da planta, constante de tempo do processo, parâmetro de ajuste definido por Skogestad. Ele determina que um valor de τ_C igual Θ fornece uma boa resposta em relação a robustez e velocidade, porém, esse valor pode ser alterado pelo usuário na busca de um controlador mais ou menos agressivo.

5.3 Projeto do Controlador de Velocidade

Como dito anteriormente, o controlador de velocidade implementado foi baseado no trabalho de Sigurd Skogestad sobre a sintonia de controladores PID utilizando a estratégia IMC [10]. A seguir, são apresentadas as seguintes etapas para a obtenção dos parâmetros do controlador: identificação e validação do modelo da planta, e obtenção dos parâmetros do controlador.

5.3.1 Identificação do Modelo da Planta

Para a identificação da planta, realizou-se primeiramente um ensaio em malha aberta onde foi aplicado um degrau de tensão de 12V na entrada do motor e recolheu-se os dados de velocidade através do encoder de quadratura. Utilizando a ferramenta *System Identification* do software MATLAB, foi escolhido o modelo de primeira ordem com atraso com a seguinte função de transferência:

$$G(s) = k\ \frac{e^{-\Theta s}}{(\tau_1 s + 1)} \tag{5.6}$$

Em que k é o ganho estático, Θ o tempo de atraso e τ_1 é a constante de tempo do processo.

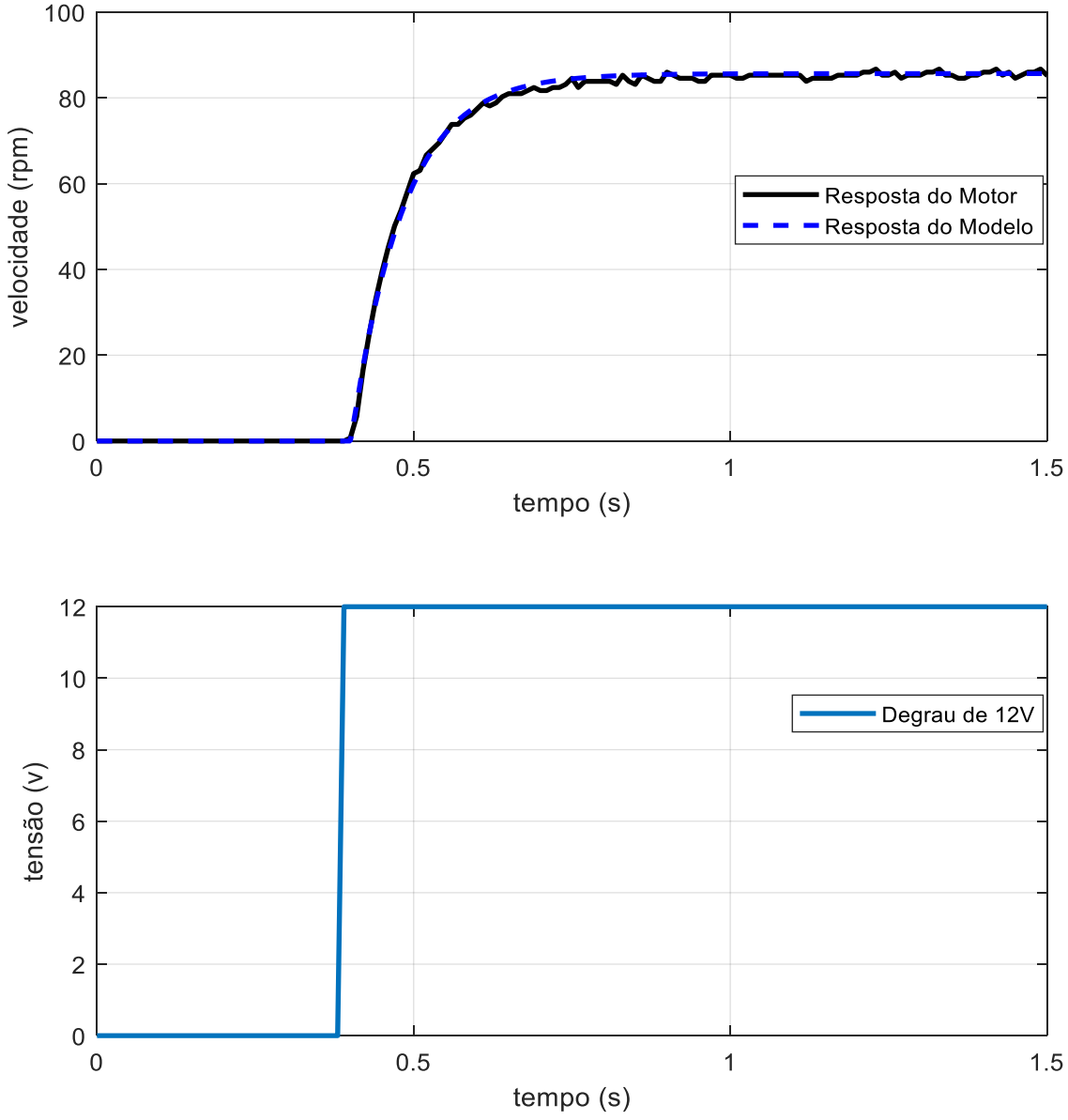
A partir dos dados extraídos do MATLAB, foi obtido o seguinte modelo no domínio da frequência:

$$G(s) = 0{,}04781\frac{e^{-0{,}0115s}}{(0{,}0818s + 1)} \tag{5.7}$$

Como forma de validação do modelo obtido, utilizou-se os mesmos dados de entrada da identificação do motor no modelo obtido através do MATLAB. Os resultados de saída e entrada de ambos foram plotados na Figura 5.1.

Na parte de cima da figura, em linha contínua e na cor preta, está a resposta do motor, e na cor azul, em linha tracejada, a resposta do modelo obtido. Na parte de baixo da mesma figura, está plotado o gráfico do degrau de entrada de 12V. De acordo com a ferramenta *System Identification,* o modelo obtido possui um *Best fits* de 97,35%. Isso pode ser observado pela proximidade da resposta do modelo com a resposta da planta no gráfico.

Figura 5.1 – Resposta da planta e do Modelo ao Ensaio de Identificação



Fonte: elaborada pelo autor.

5.3.2 Sintonia do Controlador

A partir do modelo obtido da planta, os parâmetros do controlador foram determinados segundo a Tabela 5.1, da seguinte forma:

Para:

$$\tau_C = \Theta = 0{,}0115 \tag{5.8}$$

$$K_C = \frac{1}{k}\frac{\tau_1}{\tau_C + \Theta} = 74{,}39 \quad e \tag{5.9}$$

$$\tau_I = \min\{\tau_1, 4(\tau_C + \Theta)\} = \tau_1 = 0{,}0818 \tag{5.10}$$

Com isso, a função de transferência do controlador PI no domínio da frequência obtida foi:

$$C(s) = K_C\frac{(\tau_I s + 1)}{\tau_I s} = \frac{6{,}085\ s + 74{,}39}{0{,}0818\ s} \tag{5.11}$$

Para a aplicação do controlador acima no SoC-FPGA Cyclone V, primeiramente se faz necessário a discretização do controlador e posteriormente a obtenção de sua equação de diferenças. A etapa de discretização foi feita utilizando o Método de Tustin para o tempo de amostragem do sistema de 0.01 segundos. Dessa forma, se obteve a seguinte função de transferência no tempo discreto:

$$C(z) = \frac{78{,}94\ z - 69{,}84}{z - 1} \tag{5.12}$$

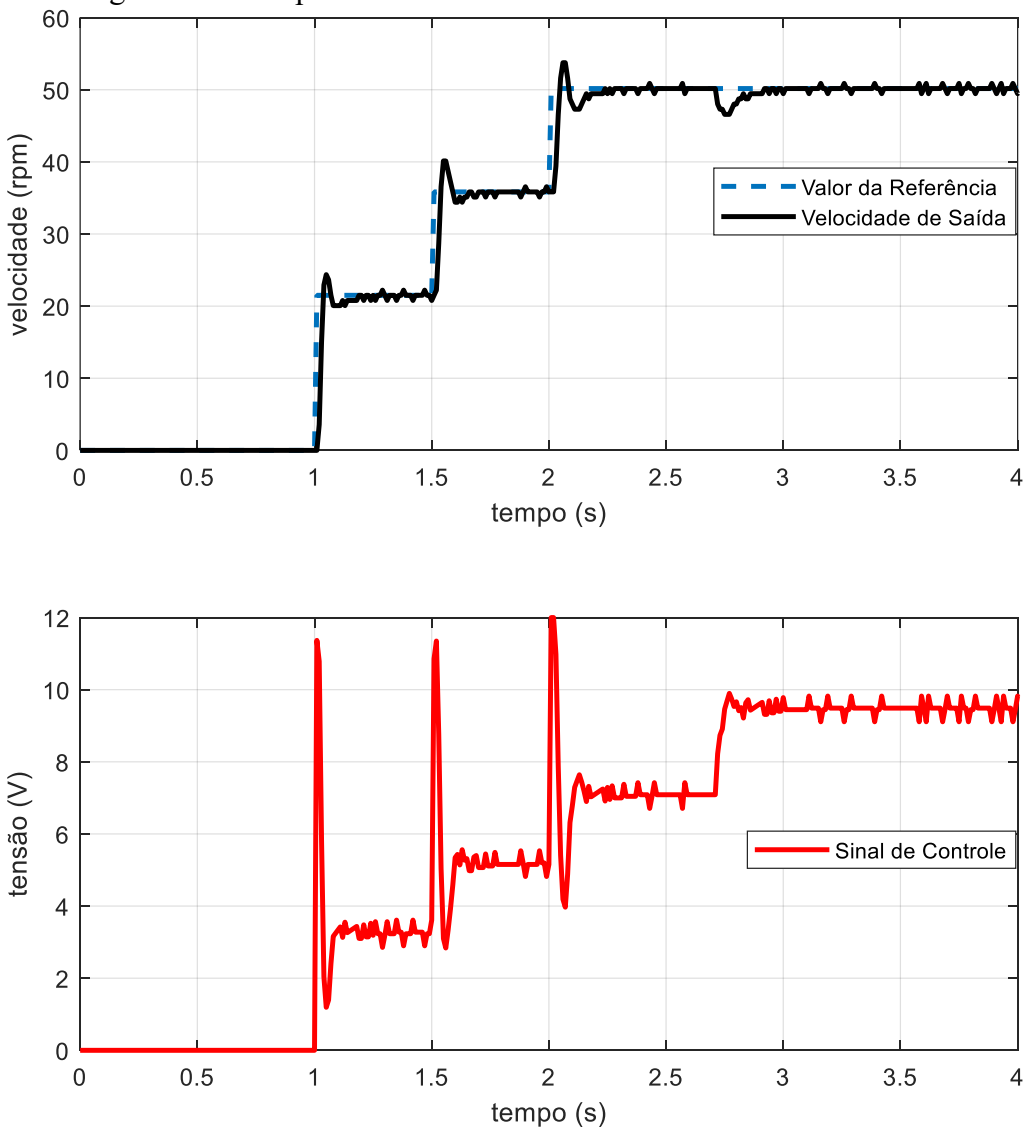
Finalmente, utilizando a Transformada Inversa de Z na função de transferência acima se obteve a seguinte equação de diferenças a ser aplicada no motor:

$$u[n] = u[n-1] + 78{,}94.e[n] - 69{,}84.e[n-1] \tag{5.13}$$

5.3.3 Resultados Experimentais

A partir da equação de diferenças do controlador descrita anteriormente, realizou-se um ensaio em malha fechada com duração de 5s. Nesse experimento, foram realizados três degraus de velocidade no motor, ocorrendo, respectivamente, em 1s, 1,5s e 2s. O primeiro degrau tem um valor de referência de 21rpm, o segundo degrau de 35rpm e o terceiro de 50rpm. Além disso, foi inserido uma perturbação de 2,4V, 2,7s após o início do ensaio. A Figura 5.2 mostra o resultado do ensaio. Na parte de cima da figura, em linha contínua e na cor preta, está o valor da velocidade em RPM do motor, e na cor azul, em linha tracejada, o valor da referência. Na parte de baixo da mesma figura, está plotado o sinal de controle.

Figura 5.2 – Resposta do Sistema em Malha Fechada



Fonte: elaborada pelo autor.

Em relação ao gráfico da resposta em malha fechada do sistema, os valores de tempo de assentamento para o critério de 5% e sobressinal são mostrados na Tabela 5.2.

Tabela 5.2 – Valores de Tempo de Acomodação e Sobressinal do ensaio

Degrau	**Tempo de Acomodação (s)**	**Sobressinal (%)**
1º	0,12	13,34
2º	0,10	12
3º	0,14	7

Fonte: elaborada pelo autor.

Sobre a perturbação inserida no sistema, a resposta apresentou um tempo de atenuação, para o critério de 5%, de 0,09s e desvio máximo 3,59rpm. Ainda sobre a perturbação, pode-se observar na Figura 5.2 que o controlador a rejeitou rapidamente. No que se refere ao sinal de controle, pode-se observar a existência de três picos, porém, em somente um deles o sinal chegou a saturar. No entanto, nota-se que essa saturação não foi suficiente para desestabilizar o sistema. Por fim, no tocante ao ruído de saída do sistema, pode se afirmar que o sinal de controle se comportou de maneira aceitável, ou seja, não precisou atuar constantemente sobre o ruído, o que o tornaria mais oscilatório.

6 CONCLUSÃO

A planta formada por um Motor CC já acoplado a um encoder se mostrou satisfatória para o teste do SoC-FPGA Cyclone V devido a sua simplicidade de configuração. O fato de o motor também vir acoplado a uma caixa de redução, só traria, aparentemente, características positivas ao sistema como o aumento de resolução do encoder de 64 pulsos por volta para 8400. Porém o que pode se observar é que ele causou um aumento significativo no tempo de atraso do sistema. Ainda, que esse aumento da parte mecânica teve de ser contornado com a introdução de um *offset* no sinal de controle, fato que alterou um pouco o desenvolvimento do *loop* de controle.

O resultado do controlador no gráfico da resposta em malha fechada mostra a eficiência da sintonia de controladores PID. Pois, com a produção de um algoritmo com poucas linhas de código e a realização de um ensaio em malha aberta foi possível chegar rapidamente a função de transferência de um controlador PI razoável. No entanto, o método de sintonia utilizado neste trabalho se mostrou apenas como um ponto de partida para outros métodos mais eficientes e que atendam a critérios de projeto mais exigentes.

A atuação SoC-FPGA Cyclone V neste trabalho indicou claramente a força dessa nova geração de FPGA's. Ao conjugar a capacidade de processamento em paralelo dos FPGA com os processadores ARM de elevado clock e uma gama diversa de periféricos, os novos SoC-FPGAs saem na frente como os mais indicados para projetos de alta complexidade e de alta demanda. Um ponto ainda limitante dos novos SoC-FPGA é o valor do chip em si. Contudo, levando em consideração que um chip pode ser programado e reprogramado diversas vezes, seu valor final ainda é bastante inferior ao preço de um ASIC, ou seja, de um circuito integrado feito para uma aplicação específica.

Como proposta para trabalhos futuros, sugere-se a utilização do kit DE1-SoC em projetos que necessitem de mais poder de processamento como o controle simultâneo de diversos atuadores através de protocolos de comunicação modernos como uma rede CAN. Além disso, a facilidade de acesso ao sistema operacional Linux embarcado sugere o desenvolvimento de um sistema supervisório do processo como um todo.

REFERÊNCIAS

[1] ASTROM, K. J.; HAGGLUND, T. **PID Controllers: Theory, Design and Tuning**. *Instrument Society of America*. 2 ed. , 1995.

[2] ASTROM, K. J.; HAGGLUND, T. **The future of PID control. Control Engineering Practice**, Vol. 9, n° 11, pp. 1163–1175, 2001.

[3] BROWN, Stephen D.; ROSE, Jonathan. **FPGA and CPLD Architectures: A Tutorial**. IEEE Design and Test of Computers. Vol. 13. No. 2, 1996.

[4] CAMPOS, M. & TEIXEIRA, H. **Controles Típicos de equipamentos e processos industriais**. São Paulo: Edigard Blucher, 2006.

[5] INTEL CORPORATION. **A Revolution in Progress: a History of Intel to Date**. Santa Barbara: Corporate Communications Department, 1984.

[6] MOROMISATO, G. D. Y. et al **A Utilização de um "Software" Livre no Ensino de Sistemas de Controle**, *International Conference on Engineering and Computer Education*. 1: 597–601, 2007.

[7] OGATA, K.: **Engenharia de Controle Moderno**: 3ª ed. São Paulo –SP, Editora LTC, 2000

[8] PALM, W. **System dynamics**. Boston, Mass: McGraw-Hill, 2 ed. 2010.

[9] SKOGESTAD, S.; GRIMHOLT, C. **Optimal PID-Control on First Order Plus Time Delay Systems & Verification of the SIMC Rules**, 2013.

[10] SKOGESTAD, S. **Simple analytic rules for model reduction and PID controller tuning**, Vol 25, No 2, pp. 85-120, 2003.