UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

DAVI ALEXANDRE PAIVA

PROJETO DE CONTROLADOR PI USANDO UM DISPOSITIVO SOC FPGA E PROCESSADOR ARM

FORTALEZA

2019

DAVI ALEXANDRE PAIVA

PROJETO DE CONTROLADOR PI USANDO UM DISPOSITIVO SOC FPGA E PROCESSADOR ARM

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica

Orientador: Prof. Dr. Paulo Peixoto Praça.

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)

P167p Paiva, Davi Alexandre.
Projeto de controlador PI usando um dispositivo SoC FPGA e processador ARM / Davi Alexandre Paiva. – 2019.
113 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. Paulo Peixoto Praça.

1. SoC FPGA. 2. Hardware. 3. Controle. 4. Linux. 5. ARM. I. Título.

CDD 621.3

DAVI ALEXANDRE PAIVA

PROJETO DE CONTROLADOR PI USANDO UM DISPOSITIVO SOC FPGA E PROCESSADOR ARM

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica

Aprovado em: ___/___/_____.

BANCA EXAMINADORA

_____
Prof. Paulo Peixoto Praça, Dr. (Orientador)
Universidade Federal do Ceará (UFC)

_____
Prof. Wilkley Bezerra Correia, Dr.
Universidade Federal do Ceará (UFC)

_____
Prof. Dalton de Araújo Honório, Dr.
Universidade Federal do Ceará (UFC)

À Deus,

Aos meus pais, Ilson e Luci,

Às minhas irmãs, Tais e Lara,

À minha linda namorada, Stephanie,

Por serem minha base e o meu amor.

AGRADECIMENTOS

À Deus, por ser o meu guia, minha referência de caráter e amor, além de ser Aquele que me traz segurança e me dar todas as condições para que eu possa me esforçar cada vez mais.

Agradecer aos meus pais, Ilson e Luci, e minhas irmãs, Taís e Lara, que sempre foram e serão a minha base, por toda a compreensão por conta da minha ausência, e mesmo quando eu não podia estar perto ou passando muito tempo junto, eles nunca deixaram de me apoiar e mostrarem seu orgulho e amor por mim, espero sempre poder retribui-los pelo tanto que eles representam, para mim.

Ao meu grande amor, Stephanie, que me ajudou e cuidou de mim, dedicando o seu melhor para que eu me sentisse cada vez mais seguro e confiante, me ensinando pacientemente cada norma da ABNT, mas principalmente com seu amor incansável, carinho e sorriso que sempre tem sido um presente para mim.

Agradecer ao professor Paulo Praça, por ter suprido com toda a instrumentação de sua alçada, nunca faltando com compreensão, paciência e disponibilidade, atributos esses admiráveis em um professor, e por fim ter me dado a oportunidade de monitoria e pesquisa.

Ao professor Fabrício, por ter cedido prontamente, laboratório, espaço, instrumentos e ajuda. Ao professor Jardel, que por mais que não me conhecesse anteriormente atendeu meu pedido de ajuda, prontamente me ajudou com toda a atenção a entender sobre algumas questões do projeto.

Por fim agradecer aos amigos de curso e de vida que muitas vezes indiretamente ou diretamente me ajudaram, a ter ideias, inspirações e soluções para desenvolver o meu trabalho de conclusão de curso. A mim mesmo, também, por todo o esforço, dedicação, por todo trabalho duro, paciência, por ter perdido tantas noites, não ter deixado de fazer o melhor.

Agradeço a todos, pois sei que por trás de qualquer coisa que eu faça há muitas pessoas me inspirando e me auxiliando cada uma de sua forma.

“Ama e faz o que quiseres. Se calares, calarás com amor;
se gritares, gritarás com amor;
se corrigires, corrigirás com amor;
se perdoares, perdoarás com amor.
Se tiveres o amor enraizado em ti,
nenhuma coisa senão o amor serão os teus frutos”.
(Santo Agostinho)

"Confie no Senhor de todo o seu coração e
não se apoie em seu próprio entendimento;
reconheça o Senhor em todos os seus caminhos,
e ele endireitará as suas veredas".
(Provérbios 3: 5-6)

RESUMO

Com o intuito de auxiliar no desenvolvimento e na pesquisa de novas implementações em FPGA, o tema desse trabalho é a integração de um dispositivo FPGA e um processador ARM por meio de um dispositivo SOC FPGA no desenvolvimento de um sistema de controle PI. Nesse trabalho é abordado um estudo dos dispositivos FPGA, linguagem HDL, microcontroladores ARM e algumas funcionalidades do sistema operacional Linux e uso de suas bibliotecas em uma aplicação de uma lei de controle. O desenvolvimento do estudo é relacionado as dificuldades envolvendo o desenvolvimento de aplicações mais complexas pelo FPGA, pois como o mesmo é sintetizado através de uma linguagem HDL, que é essencialmente voltada para a descrição de hardware, toda a lógica de programação é implementada através de elementos lógicos e modulares, tais características fazem do FPGA um dispositivo de alto poder de processamento concorrente, porém apresenta dificuldades em aplicações que requerem softwares mais complexos que geralmente são implementadas em uma linguagens de alto nível. A partir disso, esse trabalho tem como desafio a integração do processamento do FPGA com o um sistema operacional no processador ARM, para que através disso haja a abertura da possibilidade e desenvolvimento de aplicações que usem ambos os dispositivos e tenham acesso às diversas bibliotecas e funcionalidades de um sistema operacional Linux. Para a integração desses diversos dispositivos são usados três softwares principais, o *Quartus*, responsável pela programação do FPGA, o *Eclipse*, responsável pela comunicação com o sistema operacional e desenvolvimento de aplicações para o mesmo usando bibliotecas em C e por fim o *Qsys* para a integração dos dois dispositivos microcontrolador e FPGA através do mapeamento de periféricos, estabelecimento de comunicação e configuração de barramentos entre as duas arquiteturas. Com todas as ferramentas devidamente integradas é feito a programação e desenvolvimento de um sistema de controle PI testado para o controle de velocidade de um motor, tal aplicação bem simples em relação ao grande poder de processamento da ferramenta, porém bastante eficiente em demonstrar a viabilidade e a grande quantidade de possibilidades que a combinação de programação de Hardware e controladores em FPGA em conjunto com a vasta possibilidade de desenvolvimento de aplicações em um sistema operacional traz.

Palavras-Chaves: SoC FPGA. Hardware. Controle. Linux. ARM.

ABSTRACT

Intending to support the development and research of new implementations in FPGA, the theme of this work is the integration of an FPGA device and an ARM processor through a SOC FPGA device in the development of a PI control system. In this work is discussed a study of FPGA devices, HDL language, ARM microcontrollers and some functionalities of the Linux operating system and use of their libraries in an application of a control law. The development of the study is related to the difficulties involved in the development of more complex applications in FPGA, because as it is synthesized through an HDL language, which is essentially focused on the hardware description, all programming logic is implemented through elements logical and modular, such characteristics make the FPGA a high competing processing power, but presents some difficulty in applications that require more complex software that are generally implemented in a high level languages. From this, this work has the challenge of integrating FPGA processing with an operating system in the ARM processor, such that can be opened the possibility and development of applications that use both devices and access to the various libraries and features of a Linux operating system. For the integration of these different devices, three main software are used: Quartus, responsible for FPGA programming, Eclipse, responsible for communicating with the operating system and developing applications using C libraries and finally Qsys for integration of the two microcontroller and FPGA devices through peripheral mapping, communication establishment and bus configuration between the two architectures. With all the tools properly integrated the programming and development of a control system is tested PI for the speed control of a motor, such an application very simple in relation to the great power of processing of the tool, but very efficient in demonstrating the viability and the great amount of possibilities that combination of hardware programming and FPGA controllers in conjunction with the vast possibility of application development in an operating system brings.

Keywords: SoC FPGA. Hardware. Control. Linux. ARM.

LISTA DE ILUSTRAÇÕES

FIGURA 1 - ESTRUTURA DA ARQUITETURA DE UMA FPGA EM MATRIZ DE BLOCOS LÓGICOS ... 26
FIGURA 2 - FPGA EM MATRIZ SIMÉTRICA ... 27
FIGURA 3 - ABORDAGENS DE DESENVOLVIMENTO *BOTTOM-UP* E *TOP-DOWN* ... 29
FIGURA 4 - ILUSTRAÇÃO DE METODOLOGIA DE PROJETO *BOTTOM-UP* ... 31
FIGURA 5 - DIAGRAMA EM BLOCOS DO DESENVOLVIMENTO DO CHIP ARM1176JZF-S ... 36
FIGURA 6 - ILUSTRAÇÃO DO MODELO ALTERA SOC FPGA ... 43
FIGURA 7 - DISPOSITIVOS SOC FPGA COMERCIAIS ... 44
FIGURA 8 - DIAGRAMA EM BLOCOS DE UM DISPOSITIVO ALTERA SOC FPGA ... 46
FIGURA 9 - DIAGRAMA EM BLOCOS DO HPS ... 47
FIGURA 10 - DIAGRAMA DE BLOCOS DAS INTERCONEXÕES NO HPS ... 48
FIGURA 11 - KIT DE DESENVOLVIMENTO DE1-SOC FPGA E SEUS PERIFÉRICOS ... 49
FIGURA 12 - BLOCO DIAGRAMA ESQUEMÁTICO DOS PERIFÉRICOS DO DE1-SOC FPGA ... 50
FIGURA 13 – REPRESENTAÇÃO EM BLOCO DIAGRAMA DE UM SISTEMA ... 52
FIGURA 14 - BLOCO DIAGRAMA DE UM SISTEMA DE CONTROLE ... 52
FIGURA 15- EXEMPLO EM DIAGRAMA DE BLOCOS DE UM SISTEMA EM MALHA ABERTA ... 53
FIGURA 16 - EXEMPLO EM DIAGRAMA DE BLOCOS DE UM SISTEMA EM MALHA FECHADA ... 54
FIGURA 17 - DIAGRAMA EM BLOCOS ILUSTRANDO UM SISTEMA DE CONTROLE DE UM VEÍCULO 56
FIGURA 18 - DIAGRAMA EM BLOCOS DE UM SISTEMA DE CONTROLE GENÉRICO ... 57
FIGURA 19 - DIAGRAMA EM BLOCOS DE UMA MESA ROTATIVA ... 57
FIGURA 20 - EXEMPLOS DE RESPOSTA A UM SISTEMA DE RESPOSTA À UM COMANDO EM ELEVADOR ... 58
FIGURA 21 - EXEMPLO DE RESPOSTA E ANÁLISE DE PARÂMETROS DE UM SISTEMA ... 59
FIGURA 22 - DIAGRAMA EM BLOCOS DE UMA AÇÃO DE CONTROLE ON/OFF ... 60
FIGURA 23 - EXEMPLO DE UM SISTEMA DE CONTROLE DE NÍVEL DE ÁGUA ... 61
FIGURA 24 - BLOCO DIAGRAMA REPRESENTANDO UM CONTROLE DE HISTERESE DIFERENCIAL . 61
FIGURA 25 - CARACTERÍSTICAS DE UMA AÇÃO DIFERENCIAL DE UM SISTEMA A UMA MUDANÇA DE DEGRAU ... 62
FIGURA 26 - CARACTERÍSTICA DA RESPOSTA DE UMA AÇÃO PI A UM DEGRAU UNITÁRIO ... 63
FIGURA 27 - RESPOSTA DE UMA AÇÃO PD A UMA ENTRADA EM RAMPA UNITÁRIA ... 64
FIGURA 28 - RESPOSTA A AÇÃO DE CONTROLE PID A UMA ENTRADA EM RAMPA UNITÁRIA ... 64
FIGURA 29 - DISCRETIZAÇÃO DE UM CONTROLADOR ANALÓGICO ... 65
FIGURA 30 - SINAIS DISCRETIZADOS E CONTÍNUOS DO SISTEMA DA FIGURA 29 ... 66

FIGURA 31 - CÁLCULO DA INTEGRAL PELO MÉTODO DA DIFERENÇA ADIANTADA 67
FIGURA 32 - CÁLCULO DA INTEGRAL PELO MÉTODO DA DIFERENÇA AVANÇADA 68
FIGURA 33 - APROXIMAÇÃO DA INTEGRAL PELO MÉTODO DE TUSTIN 69
FIGURA 34 - PARTES ESTRUTURAIS DE UM MOTOR DE CORRENTE CONTÍNUA 72
FIGURA 35 - MODELO EQUIVALENTE DE UM CIRCUITO DE MOTOR DC 73
FIGURA 36 - ESPECIFICAÇÕES DE CARACTERÍSTICAS UMA ONDA PWM 75
FIGURA 37 - ONDA PWM PARA VÁRIOS VALORES DE *DUTY CYCLE* 76
FIGURA 38 - FORMA DE ONDA FORMADA POR DENTE DE SERRA PARA VALOR 1 DO COMPARADOR 76
FIGURA 39- FORMA DE ONDA FORMADA POR DENTE DE SERRA PARA VALOR 2 DO COMPARADOR 77
FIGURA 40 - DIAGRAMA ESQUEMÁTICO DO SISTEMA DE CONTROLE PI IMPLEMENTADO 78
FIGURA 41 - FOTO DO MOTOR DC MÓDULO 161 USADO NA IMPLEMENTAÇÃO DE CONTROLE ... 79
FIGURA 42 - ILUSTRAÇÃO DO CONJUNTO MOTOR TACOGERADOR 79
FIGURA 43 - FORMAS DE ONDA DA ENTRADA DE ONDA DO MOTOR (ONDA DE CIMA) E SAÍDA DO TACOGERADOR (ONDA DE BAIXO) PARA UMA ROTAÇÃO DE APROXIMADAMENTE 1000RPM 80
FIGURA 44 - FORMAS DE ONDA DA ENTRADA DE ONDA DO MOTOR (ONDA DE CIMA) E SAÍDA DO TACOGERADOR (ONDA DE BAIXO) PARA UMA ROTAÇÃO DE APROXIMADAMENTE 2000RPM 81
FIGURA 45 - FORMAS DE ONDA DA ENTRADA DE ONDA DO MOTOR (ONDA DE CIMA) E SAÍDA DO TACOGERADOR (ONDA DE BAIXO) PARA UMA ROTAÇÃO DE APROXIMADAMENTE 3000RPM 81
FIGURA 46 - FOTO DO MÓDULO TACHO AMP UNIT U155B, UTILIZADO NO PRESENTE PROJETO . 82
FIGURA 47 - TENSÃO RESULTANTE (ONDA DE BAIXO) DO MÓDULO *F/V CONVERTER* PRA A FREQUÊNCIA DO TACO GERADOR (ONDA DE CIMA) PARA UMA VELOCIDADE DE APROXIMADAMENTE 1000RPM..... 83
FIGURA 48 - TENSÃO RESULTANTE (ONDA DE BAIXO) DO MÓDULO F/V CONVERTER PRA A FREQUÊNCIA DO TACO GERADOR (ONDA DE CIMA) PARA UMA VELOCIDADE DE APROXIMADAMENTE 2000RPM..... 84
FIGURA 49 - TENSÃO RESULTANTE (ONDA DE BAIXO) DO MÓDULO F/V CONVERTER PRA A FREQUÊNCIA DO TACO GERADOR (ONDA DE CIMA) PARA UMA VELOCIDADE DE APROXIMADAMENTE 3000RPM..... 84

FIGURA 50 – ESQUEMÁTICO NO SIMULINK DO CIRCUITO QUE TRANSFORMA UMA ONDA SENOIDAL EM QUADRADA 85
FIGURA 51 - RESULTADO DA SIMULAÇÃO PARA O ESQUEMÁTICO DA FIGURA 51 86
FIGURA 52 - FOTO DO CIRCUITO QUE RECEBE O SINAL DO TACO GERADO E CONDICIONA PARA A ENTRADA DO FPGA 87
FIGURA 53 - FORMAS DE ONDA DO COMPARADOR PARA UMA VELOCIDADE DE 1000RPM 87
FIGURA 54 - FORMAS DE ONDA DO COMPARADOR PARA 2000RPM 88
FIGURA 55 - FORMAS DE ONDA DO COMPARADOR PARA 3000 RPM 88
FIGURA 56 - FOTO DO MÓDULO DE1-SOC FPGA USADO NO PRESENTE PROJETO 89
FIGURA 57 - DIAGRAMA DE BLOCOS DO FLUXO DE LEITURA DA VELOCIDADE DO MOTOR 90
FIGURA 58 - FLUXOGRAMA DO ALGORITMO DE LEITURA DE PERÍODO IMPLEMENTADO NO FPGA 90
FIGURA 59 - DIAGRAMA EM BLOCOS DO FLUXO DE SINAL DO PWM ATÉ A ENTRADA DO MOTOR 92
FIGURA 60 - ILUSTRAÇÃO DO MÓDULO PRÉ AMPLIFICADOR U-153 92
FIGURA 61 - FOTO DO MÓDULO U153 USADO NO PROJETO 93
FIGURA 62 - DIAGRAMA DE LIGAÇÃO INTERNA AO MÓDULO PRÉ AMPLIFICADOR 93
FIGURA 63 - SINAIS DE AMPLIFICAÇÃO DA ONDA PWM, ... 94
FIGURA 64 - FOTO DO MÓDULO DRIVER AMPLIFICADOR MOTOR U154 94
FIGURA 65 - ESQUEMA ELETRÔNICO DO MÓDULO DRIVER AMPLIFICADOR DE CORRENTE 95
FIGURA 66 - INTERFACEAMENTO ENTRE FPGA E ARM PELA PLATAFORMA QSYS TOOLS DO QUARTUS 97
FIGURA 67 - PARTE DO MANUAL ONDE SE ENCONTRA O ENDEREÇAMENTO E LOCALIZAÇÕES DOS DIVERSOS PINOS DOS DISPOSITIVOS DO SOC FPGA 98
FIGURA 68 - RESPOSTA AO DEGRAU PARA OS VALORES DE "KP" E KI" IGUAIS A 1 100
FIGURA 69 - RESPOSTA AO DEGRAU PARA OS VALORES DE "KP= 0.05" E "KI = 0.02" 101
FIGURA 70 - RESPOSTA AO DEGRAU PARA OS VALORES DE "KP= 11" E "KI = 10" 101

LISTA DE TABELAS

TABELA 1 - PROCESSADORES ARM E SUAS ARQUITETURAS 35
TABELA 2 - CÁLCULO DO VALOR DE CONTAGEM DO CONTADOR DE PERÍODO PARA DIFERENTES FREQUÊNCIAS 91
TABELA 3 - EXEMPLO DE DUAS AMOSTRAS DE VALORES DE CONTAGEM PARA DIFERENTES FREQUÊNCIAS DE SINAL DE ENTRADA 91

LISTA DE ABREVIATURAS E SIGLAS

ADCs	Analog- Digital Converter
AMBA	Advanced Microcontroller Bus Architeture
ARM	Advanced RISC Machine
AXI	Advanced eXtensible Interface
CAN	Controller Area Network
CLBs	Configurable Logic Blocks
CPLDs	Complex PLD
DMA	Direct memory Access
DMC	Dynamic Memory Controller
DSPs	Digital Signal Processing
E/S	Entrada e Saída
EMACs	Ethernet Media Acess Controllers
FIQ	Fast Interrupt
FPGAs	Field Programmable Gate Arrays
GAL	Generic Array Logic
GPIO	General Purpose I/O
HDL	Hardware Descriptor Language
HSSI	High Speed Serial Interface
I²C	Inter-Integrated Circuit
IOB	In/Out Blocks
IP	Intellectual Property
IRQ	Interruption Request
L2CC	Level 2 Cache Controller
LUT	Look- Up Table
MMC	MultiMediaCard
MMU	Memory Management Unit
MPU	Memory Protection Unit
OTG	On-The-Go
PAL	Programmable Array Logic
PD	Proporcional Derivativo
PI	Proporcional Integral
PID	Proporcional Integral Derivativo

PIT	Programmable Interval Time
PLA	Programmable Logic Array
PLDs	Programmable Logic Device
PLL	Phase-Locked Loop
PMSA	Protected Memory System Architecture
PWM	Pulse Width modulation
RTC	Real Time Clock
SB	Switch Blocks
SD	Secure Digital
SMC	Static Memory Controller
SOC	Sistem-on-Chip
SPI	Serial Peripheral Interface
SPLDs	Simple PLDs
TSC	Time Stamp Counter
ULA	Unidade Lógica Aritmética
VSMA	Virtual Memory Sistem Architecture

LISTA DE SIMBOLOS

u(t)	saída do controlador em função do tempo
U(s)	saída do controlador em função da frequência
Kp	constante de proporcionalidade
Ki	constante de integração
Ti	Tempo Integral
Td	Tempo derivativo
U(z)	Saída do controlador em transformada z
E(z)	erro aplicado a transformada z
s	variável de laplace
z	variável da transformada z
e(n)	erro da amostra atual
e(n-1)	erro de uma amostra atrás
e(n-2)	erro de duas amostras atrás
e(t)	função erro em função do tempo
E(s)	Função erro em função da frequência
$\frac{d}{dt}$	derivada
$\int$	integral
V	tensão
I	corrente
τ	Torque (conjulgado mecânico)[
ω	velocidade ângular mecânica

ϕ	fluxo magnético
E	campo elétrico
∇	Operador diferencial vetorial
v	velocidade
Q	Carga elétrica
l	comprimento
π	pi
J	Momento de Inércia
b	atrito viscoso
θ	deslocamento angular
Rf	Resistência de Campo
Ra	Resistência de Armadura
Kt	Constante taquimétrica
D	Duty cycle
Vcc	Tensão contínua

SUMÁRIO

1. INTRODUÇÃO 22

2. DISPOSITIVOS, FERRAMENTAS E PLATAFORMAS UTILIZADAS 24

2.1 DISPOSITIVO FPGA E LINGUAGEM HDL 24

2.1.1 FPGA 24

2.1.2 A arquitetura FPGA 25

2.1.3 Famílias FPGAs da Altera 27

2.2 LINGUAGEM HDL 28

2.2.1 Linguagem VHDL 29

2.2.2 Linguagem Verilog 30

2.3 MICROCONTROLADORES ARM 32

2.3.1 Modelo de negócios do processador ARM 32

2.3.2 Arquitetura 32

2.3.3 Processador 34

2.3.4 Dispositivo 35

2.3.5 Conjunto de Instruções 37

2.4 SISTEMA OPERACIONAL E KERNEL LINUX 37

2.4.1 Linux 37

2.4.2 Linux para sistemas embarcados 38

2.4.3 Linux em Tempo Real 38

2.4.4 Principais comando e bibliotecas Linux/Unix 39

2.5 DISPOSITIVO SOC-FPGA 42

2.5.1 Altera Cyclone V SoC 45

2.5.2 DE1 SOC FPGA 49

3. SISTEMAS DE CONTROLE 52

3.1 CONTROLE AUTOMÁTICO 52

3.2 PROJETO DE SISTEMA DE CONTROLE 53

3.2.1 Sistemas em Malha Aberta x Malha Fechada 53

3.2.2 Transdutores e Condicionadores de Sinal 54

3.2.3 Exemplo de Projeto de Controle 55

3.2.4 Estudo de Desempenho de sistemas controlados 58

3.3 AÇÃO DE CONTROLE 60

3.3.1 Controlador On/Off *60*
3.3.2 Ação de Controle Proporcional *62*
3.3.3 Controlador Integral Puro *63*
3.3.4 Controlador Proporcional Integral (PI) *63*
3.3.5 Controlador Proporcional Derivativo (PD) *63*
3.3.6 Controlador Proporcional Integral Derivativo (PID) *64*
3.4 TEORIA E ASPECTOS DE SISTEMAS DE CONTROLE DISCRETO 64
3.4.1 Discretização de sistemas *65*
3.4.2 Métodos de Discretização de Sistemas *66*
3.4.3 Controlador PID Digital *69*
4. CARACTERIZAÇÃO DA PLANTA 71
4.1 MOTOR DC E TACO GERADOR 71
4.1.1 Aspectos construtivos do Motor DC *71*
4.1.2 Equacionamento e Circuito Equivalente do Motor DC *72*
4.1.3 Tacogerador *74*
4.1.4 Aplicações e Características do controle por PWM *75*
4.1.5 Lógica de Implementação em programação HDL *76*
5. IMPLEMENTAÇÃO 78
5.1 COMPONENTES 78
5.1.1 Motor DC / Tacogerador *78*
5.1.2 F/V Converter *82*
5.1.3 Comparador *84*
5.1.4 SOC FPGA *88*
5.1.5 Pré Amplificador *92*
5.1.6 Driver Amplificador do Motor *94*
5.2 ALGORITMO E DETALHAMENTO DE INTERFACEAMENTO SOCFPGA 95
5.3 CONTROLADOR PI 98
5.3.1 Equação do Controlador e algoritmo de implementação *98*
5.3.2 Resultado dos Sinais Controle *100*
6. CONCLUSÃO 102
7. REFERÊNCIAS 104
APÊNDICE A: CÓDIGO EM C PARA O LINUX/ARM 106

APENDICE B: CÓDIGO EM VERILOG DO FPGA 109

1. INTRODUÇÃO

O uso de dispositivos com o uso de arquitetura FPGA tem sido cada vez mais alvo de estudos e pesquisas por suas características de síntese de hardware, pois diferente dos microcontroladores que tem essencialmente função de executar uma sequência de linhas de código de forma sequencial, o FPGA é programado para sintetizar vários blocos lógicos, tais blocos são fundamentalmente combinação de circuitos lógicos funcionando todos de modo concorrente, sem a necessidade de uma pilha de processamento. Tais características fazem do FPGA uma ferramenta muito poderosa e usada para desenvolvimento de diversos tipos de aplicações, principalmente as que requerem uma grande quantidade de uso de periféricos e processamento de dados, pois ela permite que haja uma grande expansibilidade de unidades lógicas que interagem entre si e tal quantidade não diminui o desempenho do sistema.

A arquitetura FPGA traz um vasto poder de processamento concorrente, tal arquitetura é programada em HDL que é uma linguagem de descrição de hardware, todavia a programação de dispositivos nessa arquitetura é acompanhada com uma certa complexidade para aplicações que exijam um maior nível de programação. A partir disso o objetivo desse trabalho é integrar o processamento do FPGA em conjunto com um sistema operacional no processador ARM, e exemplificar tal aplicação no desenvolvimento de um sistema de controle PI para controle de velocidade de um motor de corrente contínua.

O trabalho começa com a apresentação do dispositivo FPGA e a linguagem de programação de hardware HDL, abordando sobre os aspectos da arquitetura do dispositivo e apresentação das duas principais linguagens HDL, VHDL, Verilog, mostrando as principais características e o tipo de abordagem, construção, peculiaridades do modo de construção de hardware de cada linguagem. Logo em seguida é feita uma abordagem sobre os microcontroladores ARM, tais como arquitetura, processador, dispositivo e conjunto de instruções mostrando sua integração para base de processamento do sistema operacional LINUX, este apresentado através de suas aplicações em sistemas embarcados, sistema em tempo real, e uma explanação dos principais comandos e bibliotecas da plataforma. Baseado nessa literatura, é apresentado o dispositivo SoC FPGA que assume e integra o grande conjunto de funcionalidades advindas do FPGA e ARM através da programação HDL, C e diversas bibliotecas LINUX todos em um único chip.

Posteriormente esse trabalho busca explorar alguns conceitos da teoria de sistemas de controle e aspectos de controle discreto como a “discretização” e formas de implementação de controladores para ser usado em dispositivos de controle digital, por fim é apresentado as

características da planta controlada que é um motor CC e seus diversos ganhos, componentes, drivers e estratégias de condicionamento de sinal nos detalhes de implementação de um controlador PI. Essa implementação, busca trazer um exemplo de aplicação de alto desempenho sendo processas em um dispositivo que possua duas arquiteturas, abrindo, assim, um universo de possibilidades e soluções muito mais otimizadas através da combinação das características e funcionalidades que o uso de cada arquitetura pode trazer.

2. DISPOSITIVOS, FERRAMENTAS E PLATAFORMAS UTILIZADAS

2.1 Dispositivo FPGA e Linguagem HDL

Os PLDs (*programmable logic device)* começaram a ser introduzidos na década de 70, com o intuito de produzir circuitos de lógica combinacional que pudessem ser programados. O PLD é um chip que permite a programação do hardware (parte de implementação física do circuito) que pode ser configurado para atender à diversas especificações e funções. (PEDRONI, 2010 apud WEBER, 2016). Em essência os PLDs surgiram como uma alternativa ao uso dos chips de hardware já definido, onde para algumas aplicações preferiu-se programar o próprio hardware e implementando as funções programáveis ao invés de comprar um chip com as funções já sintetizadas de fábrica.

Inicialmente, os primeiros PLDs foram chamados de PAL (*Programmable Array Logic*) ou PLA (*Programmable Logic Array*), esses apenas implementavam circuitos combinacionais, essencialmente circuitos de operações lógicas das entradas. A estrutura posterior foi chamada de GAL (*Generic Array Logic*), nela foram adicionados circuitos lógicos na saída. Além do *flip-flop*, foram adicionadas portas lógicas e multiplexadores. Todos esses chips passaram a ser chamados de SPLDs (*Simple* PLDs), que utilizam tecnologia CMOS (*Complementary Metal-Oxide Semiconductor*) e disponibilizam elementos de memória do tipo EPROM (*Erasable Programmable Read-Only* Memory), EEPROM (*Electrically-Erasable Programmable Read-Only Memory*) e FLASH. Em meados da década de 1980, surgiram os CPLDs (*Complex* PLD) com o intuito de substituir os SPLDs e em seguida os FPGAs (*Field Programmable Gate Arrays).* Os FPGAs diferem dos CPLDs em arquitetura, tecnologia, características embutidas, tamanho, desempenho e custo (PEDRONI, 2010 apud WEBER, 2016). Eles são, de forma geral, uma matriz de blocos programáveis, ao invés de uma pilha como nos CPLDs, que possuem um número de blocos maior, com blocos menores, porém menos sofisticados.

2.1.1 FPGA

Devido ao aprimoramento das metodologias utilizadas em projetos de hardware, novas oportunidades computacionais que auxiliam no desenvolvimento de circuitos lógicos surgiram, como os dispositivos FPGAs. (TEIXEIRA,2002 apud WEBER, 2016)

O FPGA, é um dispositivo lógico programável que possui uma arquitetura baseada em blocos lógicos configuráveis chamados de CLB (*Configuration Logical Blocks*),

constituídos por portas lógicas e *flip-flop*s que visam implementar funções lógicas, também é estruturado por chamadas de blocos de entrada e saída (IOB – In/Out Blocks).

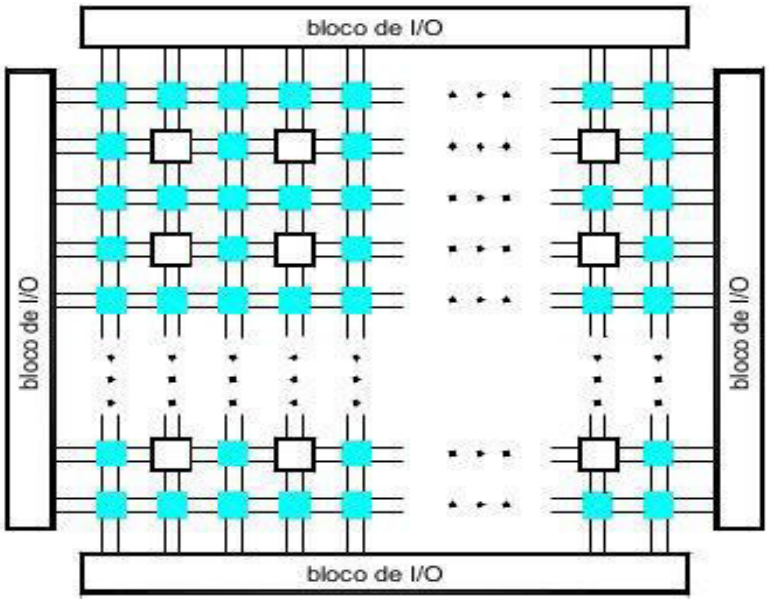
2.1.2 A arquitetura FPGA

As *Field Programmable Gate Arrays* (FPGAs), possuem uma arquitetura reconfigurável que implementa a computação de uma forma distinta dos processadores usuais, pois não fazem processamento de funções com tarefas executadas de forma sequencial ao longo de um determinado período. Estes dispositivos executam processamento em paralelo, envolvendo diversas unidades funcionais, a fim de diminuir o tempo de resposta, aumentar o desempenho de execução dos conjuntos de instruções, permitindo a customização da capacidade computacional da máquina de acordo com a aplicação. (WEBER, 2016)

As reconfigurações podem ser parciais, de forma que algumas partes do dispositivo são reconfigurados e outras mantém a configuração. Também podem ser dinâmicos, pois apresentam a possibilidade de alteração total ou parcial de um sistema, viabilizando o processo de desenvolvimento de sistemas digitais através da programação de novos hardwares em um circuito integrado digital, ao mesmo tempo que outros circuitos do mesmo componente, funcionam sem grande alteração no desempenho. (WEBER, 2016)

A estrutura de um FPGA, conforme Figura 1, é composta por blocos lógicos, blocos de I/O e chaves de interconexão. Os blocos lógicos formam uma matriz de duas dimensões e as chaves funcionam como canais que roteiam horizontalmente e verticalmente as linhas e colunas dos blocos lógicos de acordo com a necessidade dos projetos efetuados. (OLIVEIRA, 2000 apud WEBER, 2016)

Figura 1 - Estrutura da Arquitetura de uma FPGA em matriz de Blocos Lógicos



Fonte: TEIXEIRA, 2002 apud WEBER, 2016

No interior de cada bloco lógico, por exemplo o LUT (*Look- Up Table)* da Altera©, há a possibilidade de implementação de expressões lógicas através de células de armazenamento volátil que possuem a capacidade de armazenamento de valores binários. A partir a configuração da lógica combinacional em cada bloco há a possibilidade de interconexão entre vários blocos como mostra a *Figura 1,* possibilitando assim um arranjamento complexo de implementação de funções programáveis no dispositivo.

Segundo (GONÇALVES, 2005 apud RAMALHO, 2013), a arquitetura básica de um FPGA é composta por:

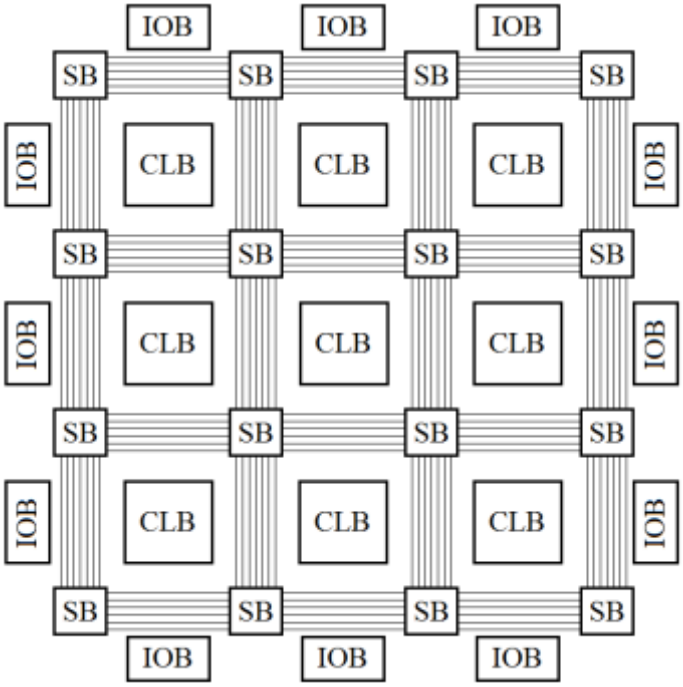
a) *configurable logic blocks* - CLBs, que contém as lógicas programáveis em linguagem de descrição;

b) *in/out blocks* - IOB, interfaces de entrada e saída para o ambiente externo;

c) *switch blocks* - SB, responsáveis pelo chaveamento do arranjo matricial entre CLBs e IOBs.

Os FPGAs mais avançados podem conter outros blocos como memória, codificadores e decodificador, entre outros. Entre os tipos de arquitetura pode-se citar a de matriz simétrica, ilustrada Figura 2**,** que possui grande flexibilidade no chaveamento de sinais.

Figura 2 - FPGA em Matriz Simétrica



Fonte: GONÇALVES, 2005 apud RAMALHO, 2013

Segundo (RAMALHO, 2013), devido à sua capacidade de chaveamento, os dispositivos FPGA podem ter processamento comparável a processadores 32 bits. Após as simulações e correções da lógica, os códigos VHDL são sintetizados em FPGA. E passam por outro período de correções, pois esses dispositivos podem ser reprogramados com facilidade.

2.1.3 Famílias FPGAs da Altera

De acordo com a (INTEL, 2019), a Intel® FPGAs oferece uma grande variedade SRAM embarcadas configuráveis, *transceivers* de alta velocidade, portas E/S (Entrada e Saída) de alta velocidade, blocos lógicos programáveis. Todo sintetizados em IP (*Intellectual Property)* combinados com ferramentas de desenvolvimento de softwares aplicados em FPGAs de baixo consumo de potência, custo.

2.1.3.1 Intel ® Agilex™ FPGAs

A família *Intel ® Agilex™* FPGAs, é produzida em tecnologia de 10nm, habilitada com aceleração e conectividade com suporte a uma grande variedade de larguras de banda e computação. Usados em aplicações de *Real- Time Actionable Inteligence, Network* (pelo suporte e capacidade de processamento em várias larguras de banda) e *Data Center (*gerenciando, organizando e processando uma grande quantidade de dados)

2.1.3.2 Intel® Stratrix® FPGAs

A família *Statrix®,* é produzida com alta performance, para aplicações envolvendo *Network, Data Science, Segurança de Redes* entre outras aplicações com máxima produtividade e poucas percas

2.1.3.3 Intel® Arria® FPGAs

A família Arria tem foco em otimização de performance e consumo em *midrange*. Os dispositivos dessa família possuem uma grande variedade de conjunto de memórias, conjunto lógicos e DSPs (*Digital Signal Processing),* combinado todos em alta velocidade com *tranceivers* 25.78 Gbps.

2.1.3.4 Intel Cyclone Inside

A família Cyclone é projetada para ter o menor consumo de potência, menor custo e grande variedade de aplicações. Muito usado em aplicações industrias de controle de motores e estabelecimento de uma central de rede local, automotivo para dar assistência de driver e monitorar consumo e elementos no veículo.

2.1.3.5 Intel ® Max ® Series

Nova família visando o menor custo de potência e grande variedade de aplicações. Que contam com uma grande variedade de (ADCs), e com todas as funções de FPGA otimizadas para muitas aplicações industriais, automotivas e de comunicação.

2.2 Linguagem HDL

As linguagens de descrição de *hardware* são amplamente utilizadas para facilitar a documentação e desenvolvimento de sistemas digitais. Essa facilidade provocou uma popularização do ambiente industrial em síntese de dispositivos de alto desempenho como CLPDs (*Complex Programmable Logic Device*) e FPGAs (*Field Programmable Gate Array*) (RAMALHO, 2013), como citados anteriormente.

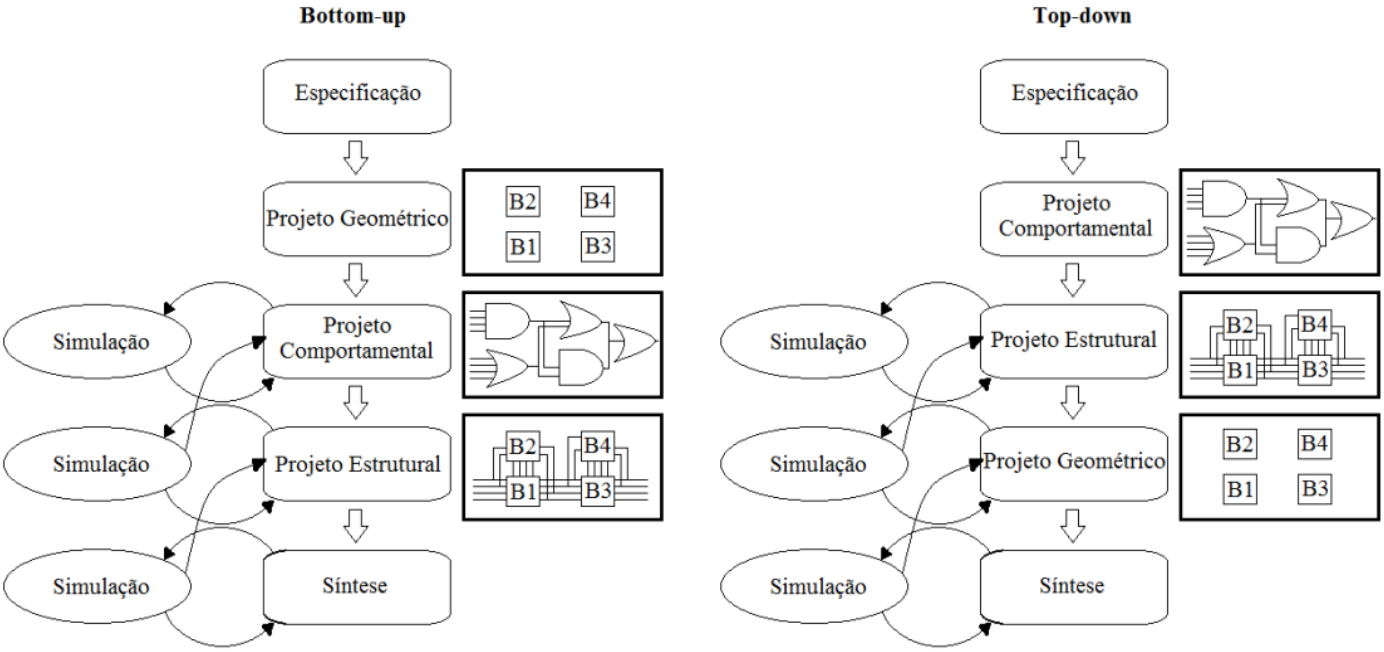
As principais linguagens de descrição de hardware utilizadas em diversas aplicações e mais comuns em *softwares* para programação de FPGAs são Verilog e VHDL. A seguir será brevemente explanado as particularidades de cada linguagem

2.2.1 Linguagem VHDL

2.2.1.1 Tipo de Abordagem

A escolha da linguagem de descrição de hardware, Verilog de abordagem "*bottom-top*", ou VHDL de abordagem "*top-down*", está relacionada às preferências, aplicações e disponibilidade de softwares. As diferenças entre as abordagens são ilustradas na Figura 3**.**

Figura 3 - Abordagens de desenvolvimento *bottom-up* e *top-down*



Fonte: GONÇALVES, 2005 apud RAMALHO, 2013

A abordagem "*top-down*" utilizada pela linguagem VHDL define que o projeto deve ser iniciado no domínio funcional ou comportamental e, etapa após etapa de simulações, ir em direção ao domínio físico de prototipagem. (RAMALHO, 2013)

De modo que se identifica o projeto principal como o topo da hierarquia e dele se tem os subprogramas que podem ser tidos como blocos lógicos complexo escritos em VHDL, esses blocos lógicos mais complexos podem se comunicar entre si através de portas de entrada e saída de sinal. Dentro de cada bloco lógico complexo encontra-se o código mais elementar onde se é instanciado as portas lógicas, *flip-flops* e instanciando blocos mais elementares de programação. Por fim dentro de cada bloco lógico se tem o uso das portas lógicas essenciais, variáveis e outros elementos de síntese de hardware.

Pela estrutura de organização de código observa-se a abordagem *top-down*, ou seja, o comportamento do sistema é tido por blocos mais complexos que são compostos pelo conjunto e integração de blocos mais elementares.

2.2.1.2 Características do Ambiente de Desenvolvimento

No ambiente de desenvolvimento em linguagens de descrição de hardware encontra-se o termo hierarquia, o qual é o ambiente onde se encontra as entidades ou blocos essenciais do programa.

As entidades identificam os projetos de hardware propostos e contém toda programação, descrição e comentários previstos em uma documentação. Todas as variáveis ou "portas", operações e componentes utilizados pela entidade devem estar instanciados nas “bibliotecas” ou nos “pacotes” utilizados no código. Caso a variação de sinais seja interna a uma entidade, então podem ser declaradas nos diversos tipos de variáveis (inteiro, real, string, entre outros). Por outro lado, a utilização das portas lógicas (Entrada ou Saída - E/S), se dá quando a variação de sinais deve ser transmitida entre duas ou mais entidades.

Dentro da documentação gerada na entidade, os “processos” são executados simultaneamente ou paralelamente. Isso porque em um hardware real, seus componentes podem funcionar simultaneamente, o que determina que as mudanças de sinais de entrada possuam atraso que se propaga para os componentes e sinais de saída. Entretanto, as instruções pertencentes a cada processo podem ser compiladas opcionalmente de forma procedural ou paralela. (RAMALHO, 2013)

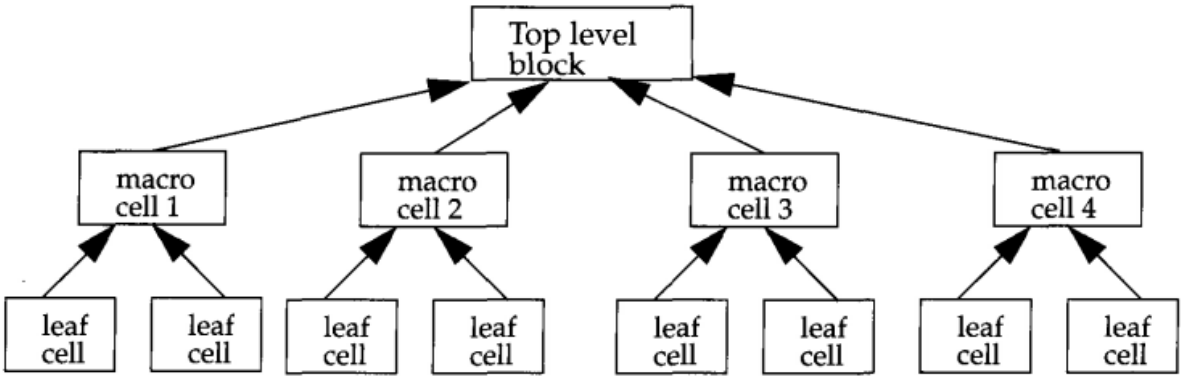
2.2.2 Linguagem Verilog

A linguagem *Verilog* foi originada em 1983 em *Gateway Design Automation.* É considerada uma linguagem de propósito geral em descrição de hardware de fácil uso e aprendizado, podendo ser comparada com a linguagem de programação em C. Permite usar diferentes níveis de abstração em um mesmo modelo, de forma que o projetista pode definir um modelo de hardware em termos de *switches, gates, RTL,* ou *“behavioral code”* (nível de abstração utilizado em testes de hardwares e ‘*test bend’)*. (PALNITKAR, 1996)

2.2.2.1 Tipo de Abordagem

A metodologia utilizada é a *bottom-up*, que primeiramente identifica-se os blocos funcionais disponível para nós, posteriormente células maiores são construídas usando esses blocos. Essas Células são então usadas como *higher-level blocks* até ser projetado o *top-level block* de fato. Essa metodologia é ilustrada na *Figura 4.*

Figura 4 - Ilustração de metodologia de projeto *Bottom-up*



Fonte: (PALNITKAR, 1996)

2.2.2.2 Características do Ambiente de Desenvolvimento

A linguagem Verilog trabalha com o conceito de módulo. Um módulo é um bloco construtivo básico que pode ser um elemento ou coleção de blocos de *lower-level.* Tipicamente, elementos são agrupados em módulos providos de funcionalidades em comum que são usados em vários lugares no projeto. Um módulo fornece as funcionalidades de necessárias de um bloco de *higher-level* através de interfaces de portas (entradas e saídas), "abstraindo" detalhes de internos de implementação. (PALNITKAR, 1996)

Verilog é tanto estruturada quanto comportamental, de forma que internamente a cada módulo podem ser combinados até quatro níveis de abstração, dependendo das necessidades do projetista. O comportamento dos módulos com o ambiente externo independe do nível de abstração utilizado, de modo que o nível de abstração pode ser mudado sem interferir a funcionalidade do mesmo no ambiente de desenvolvimento. Esses quatro níveis de abstração são: (PALNITKAR, 1996)

a) nível comportamental ou algorítmico (*Behavioral or algorithmic level),* este é o mais alto nível de abstração na linguagem Verilog HDL, pois um módulo pode ser implementado em termos de projeto de algoritmos sem detalhar conceitos de

implementação em hardware. Projetar nesse nível é similar coma programação em C;

b) nível de Fluxo de Dados (*Data Flow Level),* nesse nível o módulo é projetado pela especificação de fluxo de dados. O projetista é ciente de como é projetado o fluxo de dados entre registradores do hardware;

c) nível de Porta (*Gate Level),* o módulo é implementado em termos de portas lógicas e interconexões entres essas portas. Projetar nesse nível é similar a descrever em termos de diagrama de portas lógicas;

d) nível de Chave (*Switch level),* menor nível de abstração da linguagem. O módulo pode ser implementado em termos de chaves, nós lógicos e interconexões entre elas.

2.3 Microcontroladores ARM

2.3.1 Modelo de negócios do processador ARM

De acordo com (ARM, 2009) a ARM não fabrica o hardware do processador, ao invés disso, cria projetos de microcontroladores que são licenciados para os seus clientes que integram em *Sistem-on-Chip* (Soc) devices.

Para garantir a interoperabilidade e um modelo comum de programa em diferentes implementações, ARM define as especificações de arquitetura e como o produto ARM deve operar. Alguns parceiros licenciados também podem implementar seus próprios processadores ARM conforme as especificações da arquitetura. Para descrever o comportamento e modelo de programa de um SoC em sua totalidade, devem ser especificadas três partes:

a) arquitetura, define o comportamento que é comum para muitos projetos de processadores;

b) processador, implementação de uma arquitetura que pode ser integrada em bastante tipos diferentes de projetos;

c) dispositivo, contém o processador e componentes adicionais.

2.3.2 Arquitetura

A arquitetura ARM define como o processador ARM deve operar, isso inclui:

a) modelo de programação;

b) conjunto de instruções;

c) configurações do sistema;

d) tratamento de interrupções;

e) modelo de memória.

Novas versões de arquitetura podem ser introduzidas mudando ou adicionando algumas dessas características. Essas mudanças são normalmente compatíveis com as versões anteriores, para simplificar a migração dos novos processadores para as mais recentes arquiteturas. As versões anteriores da arquitetura ARM descrevem diferentes escolhas de implementação, como *Virtual Memory Sistem Architecture* (VSMA), baseado em MMU (*Memory Management Unit*), ou a *Protected Memory System Architecture* (PMSA), baseada na *MPU(Memory Protection Unit)*.

A arquitetura ARMv7 introduz o conceito de perfis de arquitetura, definindo versões de arquitetura destinadas para diferentes tipos de processadores em diferentes tipos de segmentos. Esses perfis são definidos como:

a) **A,** perfil de *Aplicação* define a arquitetura baseada em microprocessadores em VMSA (Arquitetura de Sistema em Memória Virtual). Voltado para processadores de alta performance, capazes de executar por completo os recursos de um SO (Sistema Operacional). Suporta tanto o conjunto de instruções ARM quanto *Thumb* (subconjunto de instruções ARM, porém comprimidas de 32 para 16 bits, reduzindo a quantidade de memória utilizado no código, diminuindo assim o hardware final);

b) **R,** perfil em *Tempo Real* define a arquitetura baseada em microprocessadores em PMSA (Arquitetura de Sistema em Memória Protegida). Voltada para sistemas que requerem um tempo determinístico e interrupções de baixa latência;

c) **M,** perfil *Microcontrolado* provê baixa latência de interrupção e acessibilidade direta da linguagem de alto-nível. Possui diferente tratamento de exceções de outros perfis, implementa a variação da PMSA e só suporta variantes do conjunto de instruções Thumb.

A ARM (*Advanced RISC Machine)* é uma família de arquiteturas que originou o uso da tecnologia RISC em aplicações comerciais, visando a simplificação das instruções buscando atingir a máxima eficiência por ciclo de trabalho.

Outras componentes dessa arquitetura são (BRUCHI, 2009, p. 16 – 17):

a) unidade de decodificação de instruções e controle lógico, decodifica as instruções ARM e Thumb e organiza a sequência de exceções e outros eventos irregulares;

b) registrador de endereço de memória, associado ao incrementador de endereço e mantém o controle da posição de PC;

c) registradores de memória de dados, controlam o fluxo de entrada e saída;

d) ULA (Unidade Lógica Aritmética), realiza operações lógicas e aritméticas requisitadas;

e) banco de registradores, 1 porta de leitura, 2 portas de escrita, portas de leitura e escrita do PC;

f) *barrel shifter*, realiza o deslocamento de uma palavra de dados em uma quantidade específica de bits.

Por fim arquiteturas mais recentes dão suporte a 7 modos de operação, listados a seguir (BRUCHI, 2009, p. 24):

a) modo usuário, execução da maioria das aplicações;

b) modo supervisor, modo em que se executa o Sistema Operacional;

c) modo de abortamento, ativado quando se tem falha de memória;

d) modo indefinido, ativado quando o processador tenta executar uma instrução que não é suportada nem pelo núcleo nem pelos coprocessadores;

e) modo de interrupção rápida (FIQ – *fast interrupt*), ativada quando o processador recebe um sinal de interrupção a partir de uma fonte designada de interrupção rápida;

f) modo de interrupção (IRQ – *interruption request*), ativado sempre que o processador recebe um sinal de interrupção;

g) modo de sistema, usado para executar certas tarefas privilegiadas do Sistema Operacional.

2.3.3 Processador

O processador é implementado para uma versão de arquitetura definida. Por exemplo, ARM926EJ-S implementa ARMv5 com a extensões, e Cortex-A9 implementa ARMv7 com a extensão de multiprocessamento. Tipicamente há muitas diferentes implementações de uma mesma arquitetura, por exemplo o ARMv4T é implementado tanto pelo ARM7TDMI quanto pelo ARM920T. A Tabela 1 mostra alguns processadores ARM com suas arquiteturas e versões.

Tabela 1 - Processadores ARM e suas arquiteturas

Implementation	***Architecture version***	***Architecture variant***
ARM 11™ MPCore™	ARMv6	ARMv6K, *Improved multiprocessing support*
ARM1156T2F-S™	ARMv6	ARMv6T2, *Thumb-2 technology*
ARM1176JZF-S™	ARMv6	1.12 ARMv6Z, ARMv6K *with Security Extensions*
Cortex-A9	ARMv7-A	
Cortex-R4	ARMv7-R	
Cortex-M3	ARMv7-M	

Fonte: (ARM, 2009)

2.3.4 Dispositivo

O dispositivo SoC geralmente incorpora o processador ARM e componentes adicionais. No momento de implementação do dispositivo geralmente podem ser incorporadas hardwares com suporte para ponto flutuante, quando opções de tamanhos de cache. De forma que diferentes dispositivos de um mesmo processador ARM podem ter diferentes tamanhos de cachê. Por exemplo, um SoC genérico deve incluir vários dos componentes seguintes: (ARM, 2009)

a) L2CC *(Level 2 Cache Controller)*, controlador de Cache Nível 2, cache responsável por implementar métodos de alta performance para processadores que possuem um alto fluxo de comunicação com a memória;

b) SMC (*Static Memory Controller)*, controlador de memória estática, é um *Advanced Microcontroller Bus Architeture (AMBA),* ou seja, uma arquitetura avançada de controle de barramento, implantada como periférico de um SOC. Geralmente é um controle de alta performance para otimiza o acesso a SRAM e os controladores de memória NAND através de uma interface de barramento implementada em AXI *(Advanced eXtensible Interface);*

c) DMC (*Dynamic Memory Controller*), controlador de memória dinâmica;

d) *bus interconnect*, conexões de barramento;

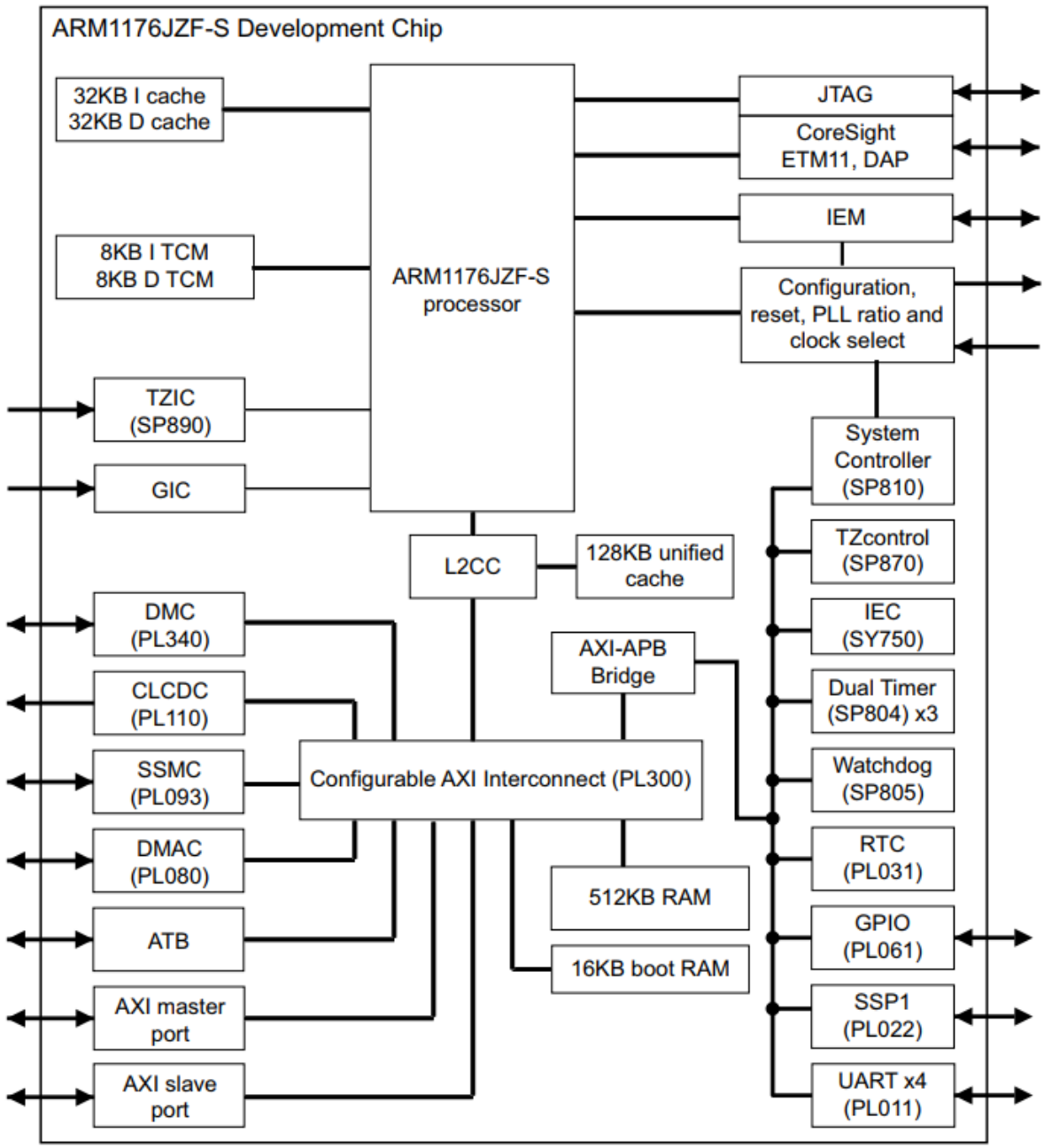
e) *interrupt controller*, controladores de interrupção;

f) timers, temporizadores;

g) *external Bus Interfaces*, interface externa de barramento.

A seguir um exemplo de SoC com processador ARM é ilustrado na Figura 5:

Figura 5 - Diagrama em blocos do desenvolvimento do chip ARM1176JZF-S



Fonte: (ARM, 2009)

2.3.5 Conjunto de Instruções

A arquitetura possui 37 registradores no total, de forma que 30 são de propósito geral, 5 dedicados para os registradores de estado do programa salvo (*sprs*), 1 dedicado para o contador de programa e o outro para o registrador de estado do programa corrente(*cprs*).

2.4 Sistema operacional e kernel linux

2.4.1 Linux

O Linux é um sistema operacional projetado para execução eficiente em várias plataformas, fornecendo interfaces de programação e de usuário compatíveis com o UNIX padrão.

De acordo com (PRADO, 2011), o Linux é uma versão gratuita do UNIX desenvolvida por Linus Torvalds, porém de desenvolvimento cooperativo, classificado como software livre (pode ser distribuído de forma gratuita) que tem se tornado cada vez mais popular pela sua flexibilidade, qualidade, poucas falhas e segurança melhor que de alguns sistemas comerciais. Esse sistema tem como partes fundamentais:

a) kernel, responsável por manter todas as abstrações importantes do sistema operacional, como memória virtual e processos;

b) bibliotecas do sistema, definição de um conjunto-padrão de funções para interação com o kernel, tais funções implementam grande parte das funcionalidades do sistema operacional;

c) utilitários do sistema, programas que realizam tarefa de gerenciamento de disco individual e especializado.

O kernel do Linux é um sistema monolítico tradicional por motivos de desempenho incorporando diversas funcionalidades para executar processos e serviços exigidos para qualificação de um sistema operacional, muitas dessas funcionalidades são fornecidas por bibliotecas do sistema, chamando os serviços do sistema operacional. O sistema também possui modularidade que garante um bom desempenho para carga e descarga de drives dinamicamente em em tempo de execução. (SILBERSCHATZ, GALVIN e GAGNE, 2016)

Segundo (LOVE, 2010 apud PRADO, 2011), o *kernel* possui alguns componentes comuns como: sistema de tratamento de interrupções, escalonador realizar a divisão de tempo de uso do processador por cada processo, gerenciador de memória, serviços de comunicação de processos e interfaces de rede.

As bibliotecas principais do Linux foram originadas pelo projeto GNU, incorporando novas funcionalidades e melhorando o desempenho dessas. Além de incorporar o compilador C GNU (gcc), diretamente no sistema operacional. De modo que o Linux é distribuído sob a GNU *General Public License (GPL)*¸ (SILBERSCHATZ, GALVIN e GAGNE, 2016)

2.4.2 Linux para sistemas embarcados

Os primeiros sistemas embarcados não eram controlados por sistemas embarcados. Os softwares eram na verdade *firmwares* projetados para acionamento direto do *hardware,* com poucas interações com o usuário, pouca atividade multitarefa e escassez de recursos que não envolvesse a programação direta do hardware. (RAGHAVAN 2006 apud PRADO 2011)

Devido à grande quantidade de bibliotecas, funções utilitárias, implementação fácil de protocolos de comunicação com a rede e um gerenciamento de memória de armazenamento e processos sendo executada à nível de kernel e não de aplicação os sistema operacional têm sido cada vez mais usado para aplicações em sistemas embarcados, com destaque ao Linux por conta da licença gratuita, suporte a grande parte das arquiteturas e comunidade de desenvolvimento livre ativa.

2.4.3 Linux em Tempo Real

De acordo com, (PICCIONI, TATIBANA e DE OLIVEIRA, 2001), o Linux é um sistema operacional que provê diversos mecanismos para manipulação em tempo real em nível de aplicação. O kernel do Linux trabalha com três relógios de *hardware,* tais como RTC (*Real Time Clock)*, TSC (*Time Stamp Counter)* e o PIT *(Programmable Interval Time),* usados para obtenção de data e hora mas também para gerar interrupções fixas programadas pelo usuário.

Uma das formas também de se trabalhar com interrupções fixas é fazendo o uso e manipulação de sinais gerados no processador, no Linux existem diversos sinais para manipulação e configuração para estouro de *timers,* interrupção de processos. Há o suporte específico para sistemas em tempo real, através de chamadas em tempo real como *tr_sigaction(), rt_sigpending(),* entre outros sinais e chamadas. (PICCIONI, TATIBANA e DE OLIVEIRA, 2001).

2.4.4 Principais comando e bibliotecas Linux/Unix

Para as especificações seguintes será usado os exemplos mais comuns de bibliotecas pertencentes ao Linux/Unix em nas versões em C de acordo com Curry (2014).

2.4.4.1 Rotinas Úteis

Muitas rotinas de uso comum são relacionadas a manipulação de *strings,* para isso tem-se o uso da biblioteca "string.h" que fornece rotinas referentes a manipulação, medição, comparação, cópia, busca e outras funções referentes. Bibliotecas relacionadas a criação de tipos e classes são fornecidas por bibliotecas de <ctype.h>, além de alocação dinâmica de memória com a biblioteca <stdlib.h>, em geral bibliotecas do tipo C, referentes a aplicações básicas dessa linguagem.

2.4.4.2 Rotinas de baixo nível em I/O

A linguagem C não provê operadores para manipulação e entradas e saídas, sendo para isso necessárias bibliotecas e rotinas de interfaceamento. As rotinas de interface direta com o sistema operacional e com o hardware são referidas como *low-level I/O interfaces,* diferente das *high-level interfaces (Standard I/O Library).*

Essas rotinas são referentes a manipulação de *files*, de forma que o identificador principal de cada *file* é o *File Descriptor,* que é um número inteiro que representa o arquivo. Há 3 *files descriptors* pré-definidos, *Standard Input* (geralmente o teclado), *Standard Output* (normalmente a tela), *Standard Error Output* (geralmente a tela também) eles são descritos pelos inteiros, 0,1,2 respectivamente.

As funções mais usuais são as de manipulações de arquivos com as funções: *open, close, read, write* que acessam diretamente os arquivos requisitados. Porém as funções de leitura e escrita importam e exportam os números em bits, ou seja, zeros e uns.

2.4.4.3 Biblioteca Standard I/O

As funções na biblioteca Standard I/O tem o intuído de abstrair as operação de leitura e escrita de variáveis e arquivos, de modo que há funções que fazem todo o processo de organização dos *buffers* e alocação das variáveis, além de facilitar também a representação dos números em binários em *int* (inteiros), *floats* (números decimais de ponto flutuante), *uint*

(inteiros não sinalizados) , *char* (caracteres) , *double* (números inteiros maiores), entre muitos outros formatos.

Outas rotinas relacionadas a essa biblioteca, são as de abrir e manipular arquivos, também de forma mais otimizada do que as rotinas de baixo nível, como a *fopen*. *fclose, fgets, fputs, fread, fwrite ,* entre outras.

Além das funções de leitura e escritas mais conhecidas, como *printf*, *scanf*, com todas as conversões de tipos numéricos, como inteiros ("*%d"* ou "*%i"* - número decimal sinalizado ,*"%o"* – número octogonal não sinalizado, *"%u"* – número decimal não sinalizados, *"%x"* – número hexadecimal não sinalizado), pontos flutuantes (*"%f"*, *"%e"*, *"%g"*), caracteres e strings (*"%c", "%s"*) etc.

2.4.4.4 Arquivos e Diretórios

Há bibliotecas e funções específicas para operar com os arquivos e diretórios armazenados ou pertencentes ao próprio sistema operacional. Nela há a manipulação de basicamente três tipos de arquivos:

a) regular, onde o arquivo se comporta simplesmente com um objeto nos arquivos do sistema onde pode ser qualquer tipo de dado que o usuário escolher de forma que o sistema operacional não o interpretara, apenas o lê ou o escreve;

b) especial, também chamado de arquivos de dispositivos, são atribuídos a eles as funções de entrada e saída com os dispositivos conectados no sistema operacional, tais como, drivers de discos, *tape drive, serial port, printer, etc).* Para acessar esses dispositivos o programa simplesmente abre o arquivo especial e o lê e escreve como se fosse um arquivo regular;

c) diretório**,** provê o mapeamento entre o nome dos arquivos e eles mesmos, impondo uma estrutura de organização de arquivos, pode conter tanto arquivos quanto diretórios.

Através da biblioteca *<unistd.h>,* tem-se a maior parte das funções relacionadas a manipulação dos arquivos, do tipo de truncamento de arquivos, modificação dos termos de acesso e permissão, criar e deletar arquivos e diretórios, renomear, ler entre outros tipos de operações.

2.4.4.5 Sinais

Os sinais são basicamente interrupções no software, eles provêm notificações assíncronas de que algo aconteceu, podem ser problemas que apareceram, usuários ou processos que requisitam que o processador trate algum pedido fora de operação normal. Esses sinais podem ter tanto causas diretas vindas do hardware, quanto causas puramente de software.

Para cada sinal definido no sistema operacional há quatro formas de lidar com o seu disparo, o mesmo pode ser:

a) ignorado, dizendo então para o sistema operacional imediatamente descartar o sinal sem entrar em um processo, ou "rotina de interrupção";

b) bloqueado ou **"***held",* quando o sinal é bloqueado ele não entra no processo imediatamente, ao invés disso, o seu pedido é levado a uma fila de processos pendentes. Se o processo corrente não bloquear o sinal ele será tratado naquele momento. Sinais bloqueados são geralmente usados em "seções críticas" onde os processos não podem ser interrompidos;

c) preso ou capturado**,** o processo pode dizer para o sistema operacional que independente do momento no qual o sinal é entregue uma função definida pelo usuário de "*signal handler"* dever ser chamada. Após a rotina ser tratada o sistema operacional volta a funcionar de onde parou;

d) *default*, de modo que se não for especificado, o sinal ele será tratado quando terminar os processos em execução no sistema operacional.

A versão 7 do UNIX provê 15 diferentes tipos de sinais, caracterizados de acordo com sua operação, que pode ser em comunicação entre processos, conexões networking, há também sinais de propósito específico que são os sinais usados em programação de sistemas em tempo real. Um dos principais sinais são:

a) *SIGINT* (Interrupção), sinal entregue ao processo quando o usuário pressiona o botão de interrupção, (geralmente "*ctrl-c"*) no teclado;

b) *SIGQUIT* (Abandono*),* sinal entregue ao processo quando o usuário pressiona o botão de sair, (geralmente "*ctrl-\"*, ou "*ctrl-d"*) no teclado;

c) *SIGTRAP (Trace/breakpoint trap),* sinal utilizado em situações de depuração do programa, ou seja, são para interrupções a cada linha de comando;

d) *SIGFPE* (*Arithmetic exception),* sinal usado para sinalizar processos com operações aritméticas ilegais, como divisões por zero e overflow de pontos flutuantes;

e) *SIGALARM* **(***Alarm clock),* sinal entregue ao processo quando um alarme é programado, ou o sistema de contagem "*settimer"* termina, posto em funcionamento através da diretiva "*alarm()"*;

f) *SIGUSR1* **(***User-definied sinal one),* sinal que pode ser usado pra qualquer propósito de comunicação entre processos definido pelo usuário;

g) *SIGUSR2* **(***User-definied sinal two),* sinal que pode ser usado pra qualquer propósito de comunicação entre processos definido pelo usuário;

h) *SIGVTALARM* **(***Virtual timer expiration),* sinal entregue ao processo quando o tempo agendado no timer virtual através do *setitimer* é expirado, operação no modo "*itimer_virtual"*;

i) *SIGPROF* (*Profiling timer expiration),* sinal entregue ao processo quando o "*profiling timer alarm"* é agendado através do *setitimer* é expirado, operação no modo "*itimer_prof"*.

2.4.4.6 Network

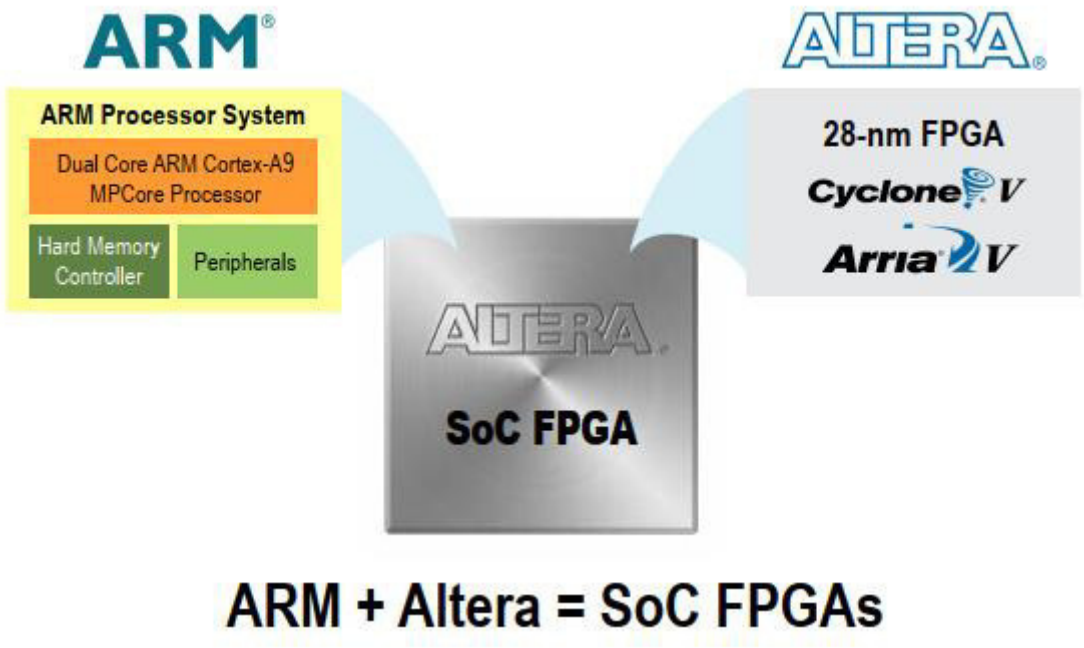
Uma das grandes funcionalidades de um sistema em Linux é a grande quantidade de bibliotecas e funções relacionadas a conectividade com a internet e capacidade de conexão com a rede, respeitando seus protocolos e funções básicas.

O Linux possui bibliotecas e funções dedicadas ao desenvolvimento de aplicações envolvendo networking, conexão com a rede, funções relacionadas a manipulação de *hosts*, *address*, serviços e enumeração de portas, criação e manipulação de sockets, manipulação se servidores e transferência de dados. Além de outras funcionalidades do universo de redes de computadores.

2.5 Dispositivo SOC-FPGA

O dispositivo SoC FPGA (Figura 6) integra um processador e uma arquitetura FPGA em um único dispositivo integrando o alto nível de funcionalidades de gerenciamento no processador com operações em tempo real, extremo processamento de dados, funções de interfaceamento do FPGA tudo em um único dispositivo. Provê, um alto nível de integração, baixo consumo e uma alta largura de banda de comunicação entre o processador e o FPGA. Esse dispositivo também contém um rico set de periféricos, chips de memória, matriz FPGA programável e alta velocidade de "*transceivers".* (ALTERA CORPORATION, 2014)

Figura 6 - Ilustração do modelo Altera SoC FPGA



Fonte: (ALTERA CORPORATION, 2014)

Na Figura 7, tem-se o exemplo de três SoC FPGAs disponíveis no mercado. Os processadores desses dispositivos são totalmente dedicados do tipo Hard Processor diferente dos Soft Processors que são processadores sintetizados por código no FPGA, ou seja, um exemplo de processador "*soft*" com memória, controladores de memória, barramentos e periféricos através de elementos lógicos do FPGA tem-se o NIOS II. Nesse presente trabalho desenvolve-se o projeto no SOCFPGA da Altera, o qual será agora tópico de estudo. Porém segue um exemplo de outros dispositivos integrados:

Figura 7 - Dispositivos SoC FPGA comerciais

	Altera SoC	Xilinx Zynq 7000 EPP	Microsemi SmartFusion2
Processor	ARM Cortex-A9	ARM Cortex-A9	ARM Cortex-M3
Processor Class	Application processor	Application processor	Microcontroller
Single or Dual Core	Single or Dual	Dual	Single
Processor Max. Frequency	1.05 GHz	1.0 GHz	166 MHz
L1 Cache	Data: 32 KB Instruction: 32 KB	Data: 32 KB Instruction: 32 KB	No data cache Instruction: 8 KB
L2 Cache	Unified: 512 KB, with Error Correction Code	Unified: 512 KB	Not Available
Memory Management Unit (MMU)	Yes	Yes	Yes
Floating Point Unit/NEON Multimedia Engine	Yes	Yes	Not available
Acceleration Coherency Port (ACP)	Yes	Yes	Not available
Interrupt Controller	Generic (GIC)	Generic (GIC)	Nested, vectored (NVIC)
On-Chip Processor RAM	64 KB, with ECC	256 KB, no ECC	64 KB, no ECC
Direct Memory Access Controller	8-channel ARM DMA330 32 peripheral requests (FPGA + hard processor system	8-channel ARM DMA330 4 peripheral requests (FPGA only)	1-channel HPDMA 4 requests
External Memory Controller	Yes	Yes	Yes
Memory Types Supported	LPDDR2, DDR2, DDR3L, DDR3	LPDDR2, DDR2, DDR3L, DDR3	LPDDR, DDR2, DDR3
External Memory ECC	16 bit, 32 bit	16-bit	8 bit, 16 bit, 32 bit
External Memory Bus Max. Frequency	400 MHz (Cyclone V SoC), 533 MHz (Arria V SoC)	667 MHz	333 MHz
Processor Peripherals	1x Quad SPI controller 1x NAND controller 2x 10/100/1G Ethernet controller 2x USB 2.0 On the Go (OTG) controller 1x SD/MMC/SDIO controller 2x UART 4x I²C controller 2x CAN controller 2x SPI master, 2x SPI slave controller 4x 32 bit general-purpose timers 2x 32 bit watchdog timers	1x Quad SPI controller 1x static memory controller (NAND, NOR, or SSRAM) 2x 10/100/1G Ethernet controller 2x USB 2.0 OTG controller 2x SD/SDIO controller 2x UART 2x I²C controller 2x CAN controller 2x SPI controllers (master or slave) 2x 16 bit triple-mode timer/counters 1x 24 bit watchdog timer	1x 10/100/1G Ethernet controller 2x USB 2.0 OTG controller 2x UART 2x I²C controller 1x CAN controller 2x SPI 2x general-purpose timers 1x watchdog timer 1x real-time clock (RTC)
FPGA Fabric	Cyclone V, Arria V	Artix-7, Kintex-7	Fusion2
FPGA Logic Density Range	25 K to 462 K LE	28K to 444 K LC	6 K to 146 K LE
Hardened Memory Controllers in FPGA	Up to 3, with ECC	Not available	Not available
High-speed Transceivers	Available at all densities	Higher-density devices only	Higher-density devices only
Analog Mixed Signal (AMS)	Not available	2 x 12-bit, 1 MSPS analog-to-digital converters (ADCs)	Not available
Boot Sequence	Processor first, FPGA first, or both simultaneous	Processor first	Processor first

Fonte: (ALTERA CORPORATION, 2014)

Os elementos principais da tecnologia *Intel*© SOC FPGA são: (INTEL CORPORATION, 2017)

a) ARM–*BASED Hard Processor System*, tipo de HPS que consiste na integração do processador de múltiplos núcleo ARM CORTEX com uma grande variedade de periféricos, e controladores de memória multiportas com todas essas funcionalidades e controle compartilhados com o FPGA. De modo que, o HPS já é sintetizado em sua fabricação com muitos periféricos em hardware de controladores de memória, *timers* entre outros;

b) *High-Speed Interconnect,* interconexões específicas entre HPS e FPGA providas em um dispositivo SOC FPGA que são de um nível tal de performance que não podem ser atingidas em um sistema comum de dois chips. Por exemplo no Arrial V SoC provê uma velocidade de transmissão por banda de mais de 125 Gbps.

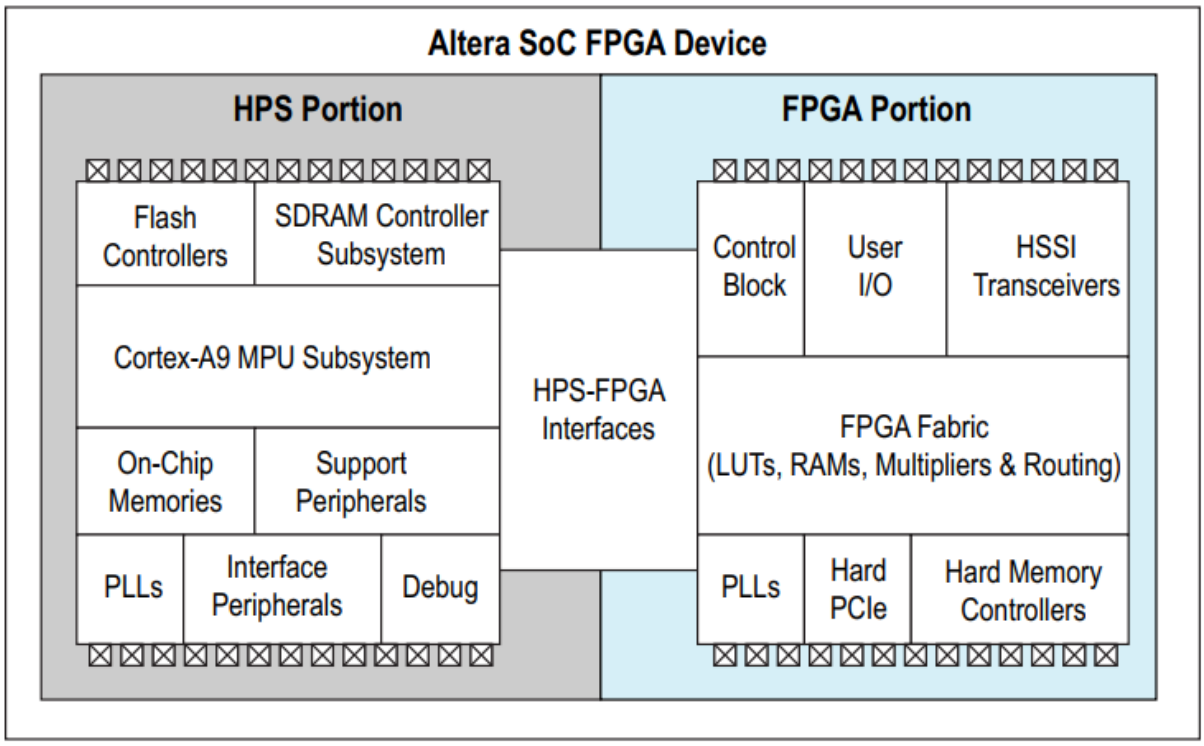
c) A Intel também trabalha pra a compatibilidade com a comunidade de LINUX de modo que provê suporte para U-Boot, LTSI kernel e Yocto Projetct layer, e outros kernels do Linux, além de também suportar outros sistemas operacionais como Wind River VxWorks, Green Hills Software Integrity RTOS and Micrium µC/OS-II and µC/OS-III dependendo da placa usada de acordo com o suporte intel.

A dispositivo usado nesse presente trabalho é o Cyclone V Soc da Altera.

2.5.1 Altera Cyclone V SoC

O dispositivo Cyclone V é um singular Sistema em Chip (SoC) que consiste em duas partes distintas, a parte do HPS (*Hard Process System)* e a parte do FPGA (*Fiel Programable Gate Arry),* como mostrado na Figura 8. (ALTERA, 2012)

Figura 8 - Diagrama em blocos de um dispositivo Altera Soc Fpga



Fonte: (ALTERA, 2012)

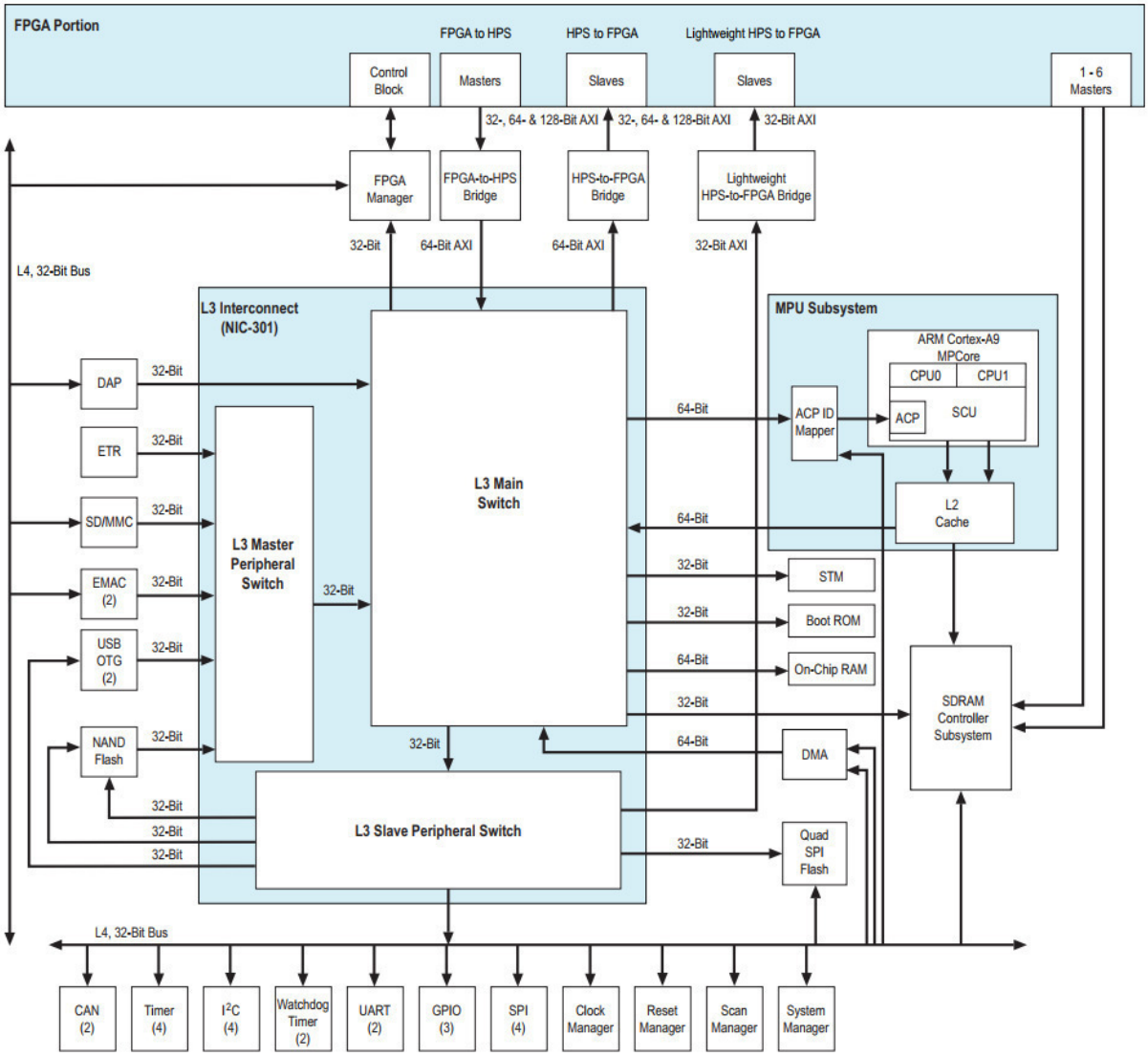
O HPS contém uma unidade de microprocessador (MPU) com um ou dois núcleos processadores ARM® *Cortex* ™ - A9 *MPCore*, controladores de memória flash, interconexões SDRAM L3, memória em chip (on -chip), periféricos de suporte, interfaces dos periféricos, capacidade de debug, e PLLs (*phase-locked loop).* O processador dual HPS suporta multiprocessamento simétrico (SMP) e assimétrico (AMP).

A porção do FPGA contém *FPGA Fabric,* bloco de controle (CB), PLLs (*phase-locked loop),* e dependendo da versão do dispositivo, pode conter, Transceiver HSSI (*High Speed Serial Interface)*, controlador *hard* PCI Express®, e *hard memory controller.*

As porções do HPS e FPGA são distintas e podem ser usadas separadamente independentes ou em conjunto, de modo que o boot do HPS pode ser feito de múltiplas fontes incluindo do *FPGA fabric e* dispositivo externo flash, e por conseguinte o FPGA também pode ser programada tanto pelo HPS quanto por qualquer outro fonte externa suportada pelo dispositivo.

A seguir um bloco diagrama esquemático (Figura 9) que ilustra o HPS.

Figura 9 - Diagrama em blocos do HPS



Fonte: (ALTERA, 2012)

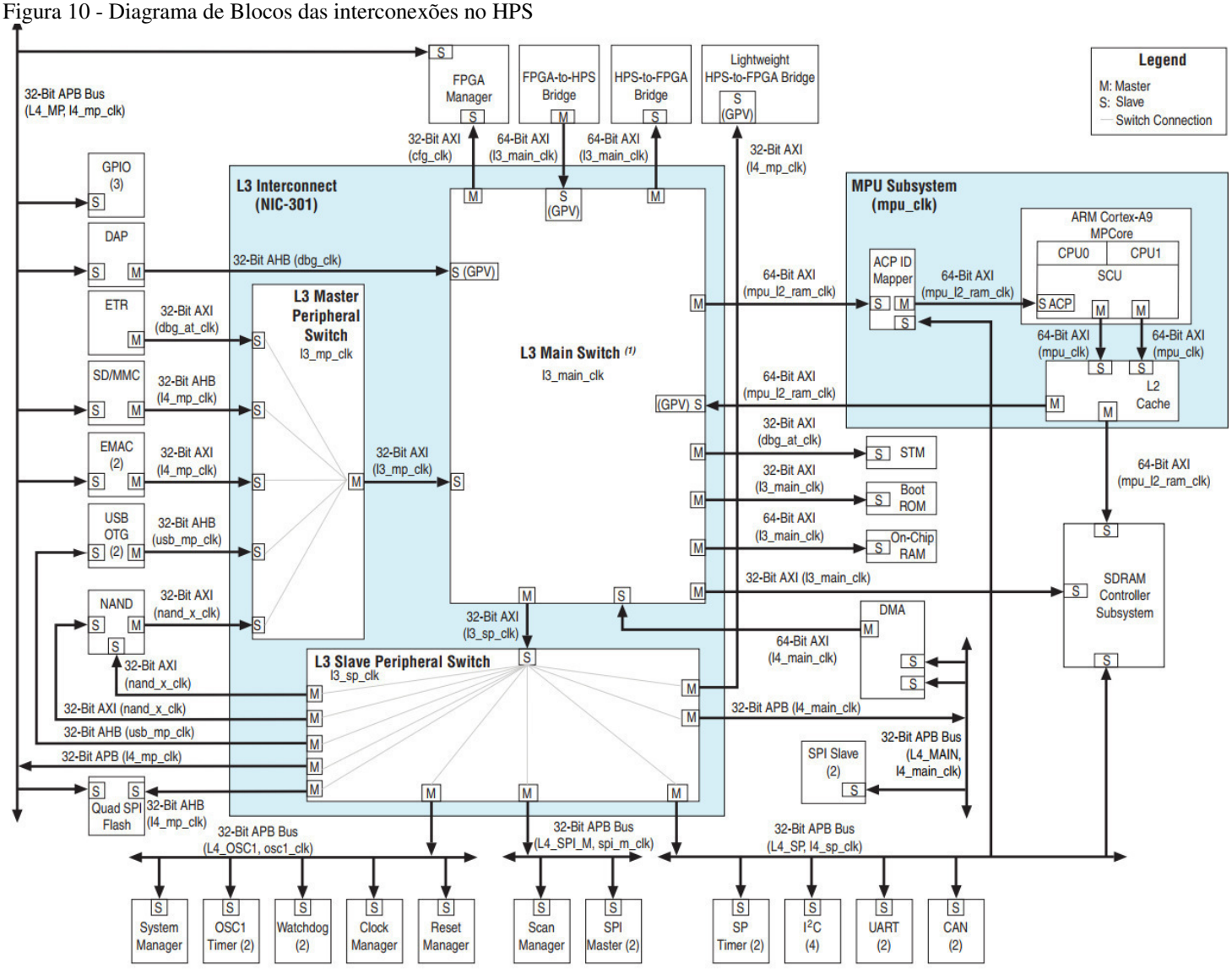
O HPS-FPGA possui uma grande variedade de canais de intercomunicação, baseados nas interconexões level 3 (L3) e level 4 (L4) barramento de periféricos que são implementados com o ARM® Corel Ling™ Networking Interconnect (NIC-301). O NIC-301 provê os fundamentos de interconexões de HPS em alta performance para ARM baseado nos protocolos *Advanced Microcontrollers Bus Architecture (AMBA®), Advanced eXtensible Interface(AXI ™), Advanced High Perfomance Bus (AHB™), e Advanced Peripheral Bus (APB™).*

A interconexão *L3* implementa multicamadas, arquitetura que suporta múltiplos e simultâneas transações entre mestres e escravo. A interconexão provê cinco barramentos independentes do tipo L4 para registradores de controle de acesso e status (CSRs) de periféricos, gerenciadores, e controladores de memória.

De forma simplificada pode-se dizer que as conexões do tipo L3 são geralmente para interconexões entre os diversos controladores e memorias no HPS e as conexões do tipo L4 provê as conexões com os periféricos, todas em alto padrão de performance. As principais interfaces entre os diversos elementos no HPS podem ser exemplificadas pela matriz de interconexões de modo que conectados ao:

Um bloco diagrama esquemático é mostrado na figura 10 para ilustração das interconexões no HPS. (ALTERA, 2012)

Figura 10 - Diagrama de Blocos das interconexões no HPS

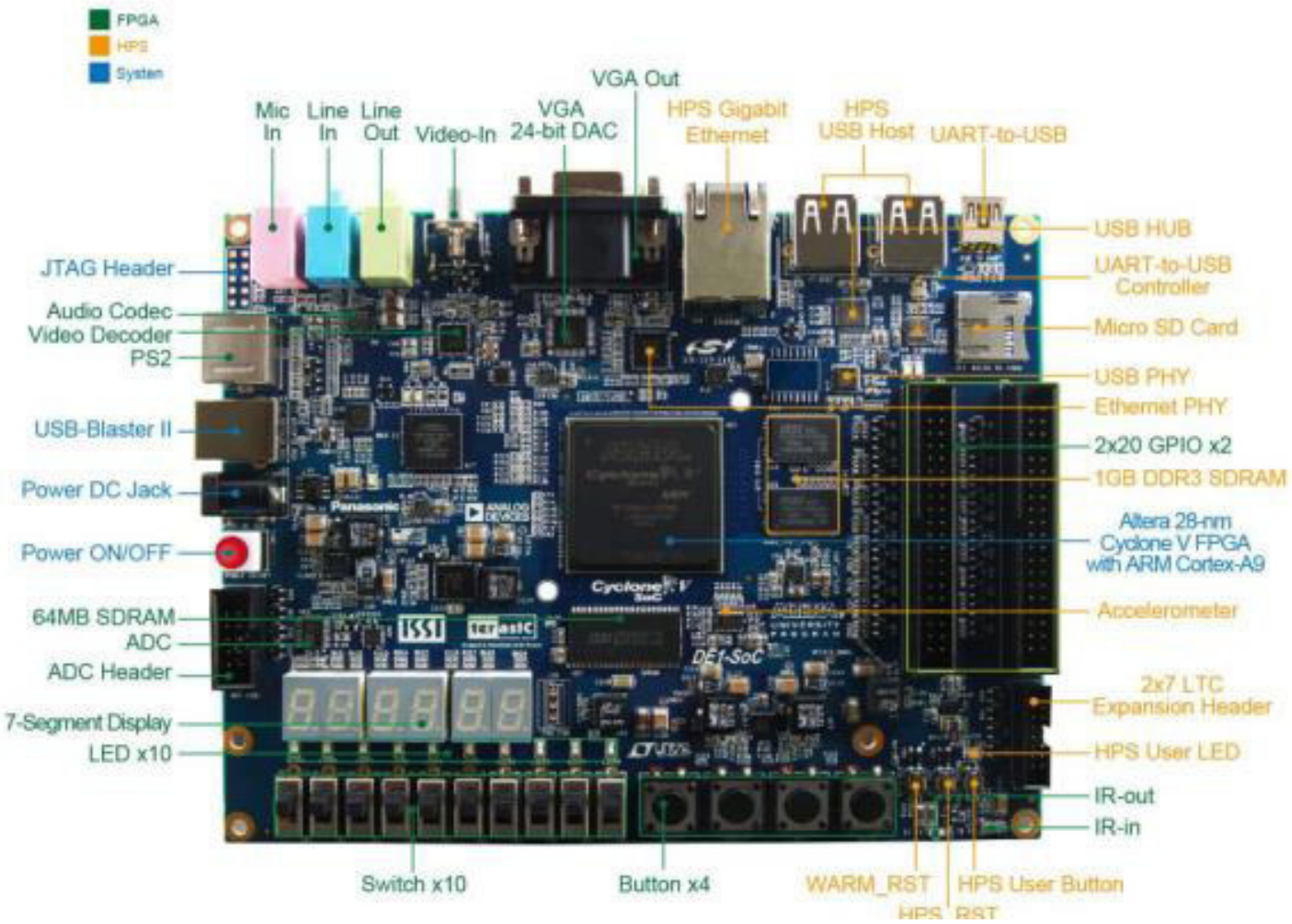


Fonte: (ALTERA, 2012)

2.5.2 DE1 SOC FPGA

O kit de desenvolvimento utilizado nesse presente trabalho foi o DE1-SoC *Developement Kit* (Figura 11) construído ao redor da arquitetura do SoC FPGA de integração entre o FPGA Cyclone V e o ARM Cortex A9.

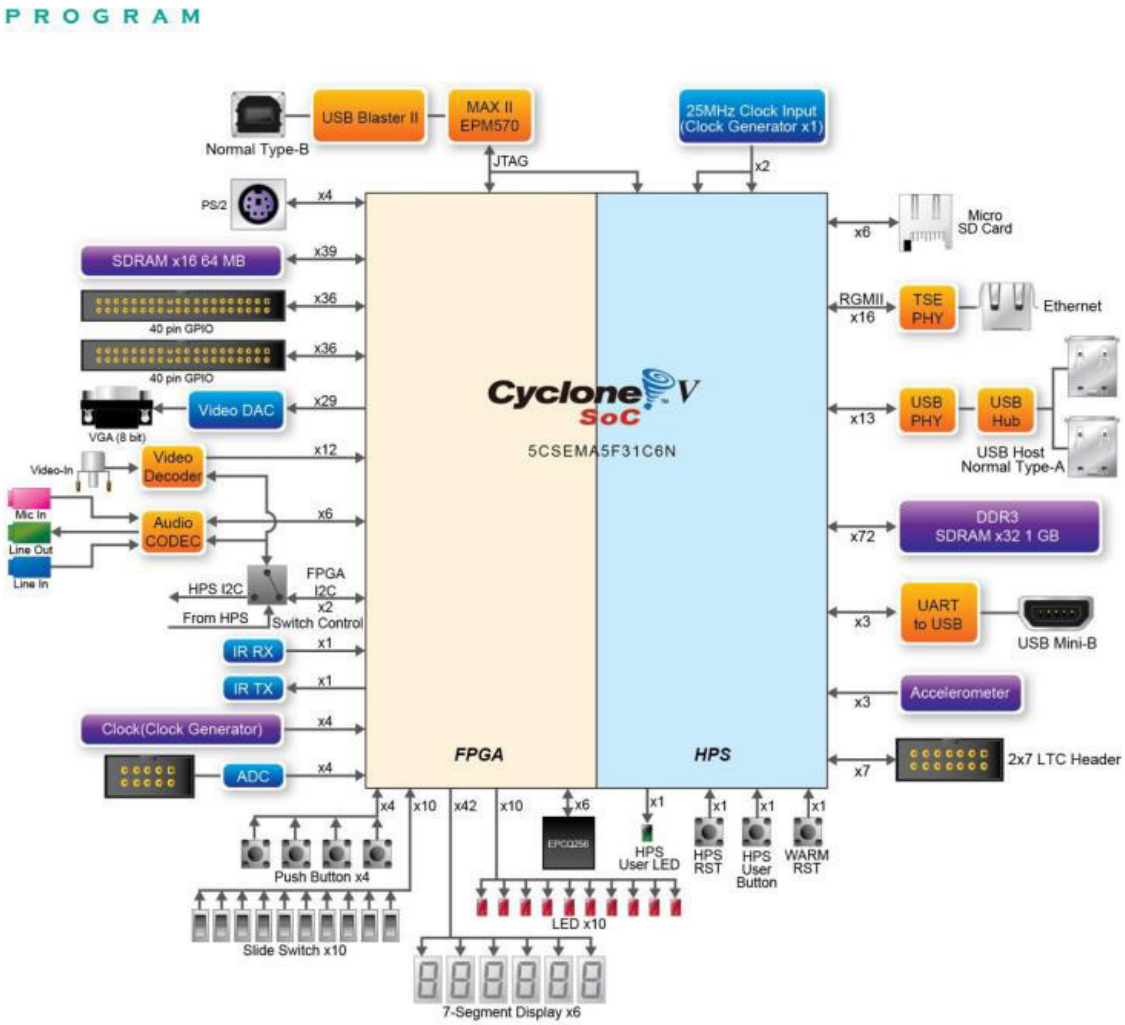
Figura 11 - Kit de desenvolvimento DE1-SoC FPGA e seus periféricos



Fonte: (TERASIC TECHONOLOGIES, 2015)

O kit possui vários periféricos do HPS e do FPGA como mostrados (Figura 12).

Figura 12 - Bloco diagrama esquemático dos periféricos do DE1-SOC FPGA



Fonte: (TERASIC TECHONOLOGIES, 2015)

Em relação a parte de hardware da placa da parte do FPGA temos:

a) dispositivo FPGA, Altera Cyclone® V SE 5CSEMA5F31C6N;

b) dispositivo de configuração serial da Altera EPCS128;

c) programador, *USB-Blaster II* e *JTAG Mode;*

d) memória SDRAM com barramento de dados de 16bits e tamanho de 64MB;

e) 4 *push-buttons*, 10 *slide switches*, 10 *LEDs vermelhors;*

f) 4 *clocks* de 50Mhz do gerador de *clock;*

g) audio *codec* 24 bits com *line-in, line-out, and microphone-in jacks*;

h) *VGA DAC* (triplo DACs de alta velocidade em 8-bit) com conector de saída VGA;

i) decodificador de TV (NTSC/PAL/SECAM) e conector de entrada TV-in;

j) conector de mouse e teclado do tipo PS2;

k) emissor e resceptor IR (*infra-red);*
l) 40 pinos de expansão com diodo de proteção;
m) conversor A/D, 4-pin *SPI interface* com o FPGA

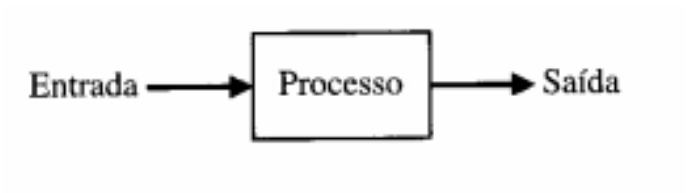
Em relação a parte de hardware da placa da parte do HPS temos:

a) processador ARM Cortex-A9 Dual -core;
b) memória SDRAM DDR3 de 1GB com barramento de dados de 32 bits;
c) 1 Gigabit *Ethernet PHY,* com conector RJ45;
d) 2 ports USB Host com conector tipo A;
e) *socket cartão micro SD;*
f) acelerômetro com interface I2C e interrupção;
g) UART para USB com conector mini B;
h) Botões de *warm reset* e *cold reset;*
i) 1 botão e led para usuário;
j) *header* de expansão LTX 2x7.

3. SISTEMAS DE CONTROLE

Pode se observa na (Figura 13) um sistema qualquer sem controle, onde se tem uma entrada e ele reponde com uma saída característica do processo (sistema). (DORF e BISHOP, 2001)

Figura 13 – Representação em bloco diagrama de um sistema

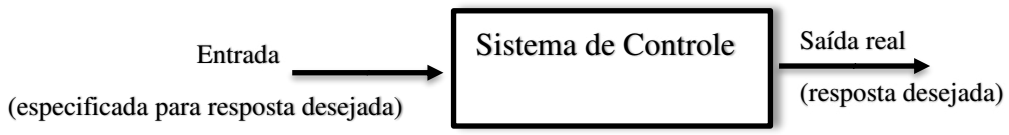


Fonte: (DORF e BISHOP, 2001)

> Um sistema de controle consiste em subsistemas e processos (ou plantas) construídos com o objetivo de se obter uma saída desejada com um desempenho desejado, dada uma entrada especificada. (NISE, 2012)

De forma simplifica um sistema de controle pode ser ilustrado pela Figura 14, em que se aplica uma entrada (excitação) ao sistema de acordo com uma reposta deseja, então o sistema responde com uma saída real, que deve-se aplicar os estudos e técnicas de controle para que essa resposta real seja igual a resposta deseja.

Figura 14 - Bloco diagrama de um sistema de controle



Fonte: Próprio Autor

3.1 Controle Automático

De acordo com (DORF e BISHOP, 2001) os sistemas de controle são bastante usados em várias áreas da engenharia, medicina, economia, automobilística, robótica etc, em que cada vez mais sistemas são automatizados para que tenham suas respostas de acordo com o esperado e projetado para diversas aplicações. Os exemplos são abrangentes que vão desde pequenos motores em impressoras até grandes máquinas elétricas de suprimento de energia do sistema elétrico interligado nacional, de modo que os sistemas de controle são cada vez mais necessários em equipamentos e máquinas relacionadas à automação industrial, em que há máquinas de diversas funcionalidades em que a precisão, exatidão, velocidade e robustez

fatores muito importantes. De modo que o estudo e desenvolvimento de técnicas de controle são grandes áreas de estudo e desenvolvimento em engenharia.

3.2 Projeto de Sistema de Controle

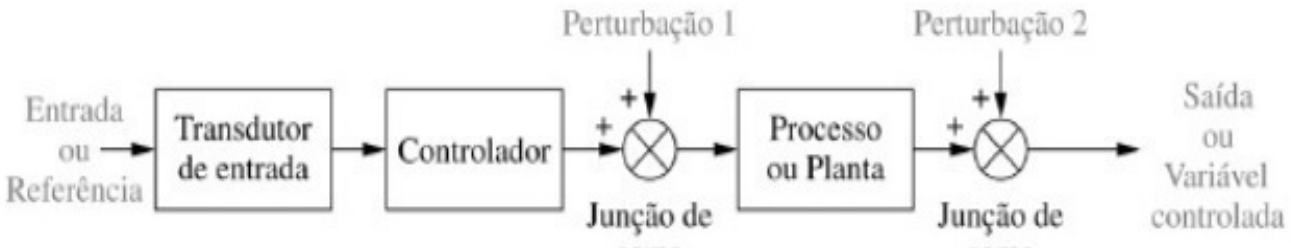
Para um projeto de um sistema de controle são fundamentais alguns conceitos, definições e fundamentos citados a seguir:

3.2.1 Sistemas em Malha Aberta x Malha Fechada

De acordo com (NISE, 2012) um sistema em malha aberta começa com um subsistema chamado *transdutor de entrada* o qual converte a forma da entrada para aquela utilizada pelo controlador que por sua vez aciona a planta ou processo o qual se pretende atuar, como ilustrado (Figura 15). Já um sistema em malha fechada (Figura 16) ele tem acesso através de um sensor ao valor de saída da planta, tal valor é subtraído da entrada formando um sinal de erro constituído pela diferença entre a saída real e a saída desejada.

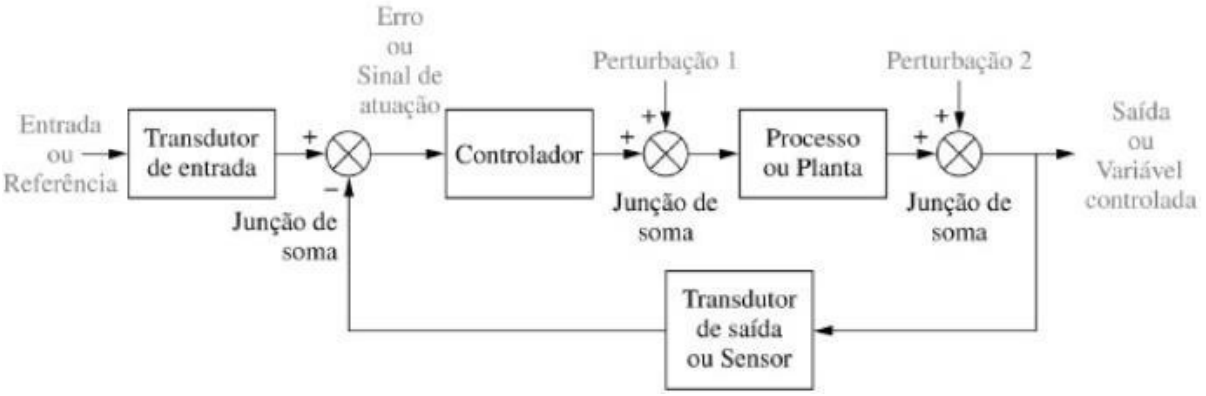
Portanto, através das ilustrações seguintes percebe-se que em um sistema em malha aberta onde se tem uma entrada porém a saída não é monitorada pelo próprio sistema a saída real do sistema pode sofrer perturbações e ele não responda da forma como esperada, tais perturbações podem ser compensadas se o sistema possuir realimentação ou retroação que é justamente o acesso ao valor de saída para poder ser comparado com o valor da referência, de modo que o controlador tem acesso ao erro, ou seja a diferença entre o que se espera (referência) e o que se tem de fato (saída), estratégia essa denominada de realimentação negativa, que é usada por conta de aspectos de estabilidade no sistema de controle.

Figura 15 - Exemplo em diagrama de blocos de um sistema em malha aberta



Fonte: (NISE, 2012)

Figura 16 - Exemplo em diagrama de blocos de um sistema em malha fechada



Fonte: (NISE, 2012)

3.2.2 Transdutores e Condicionadores de Sinal

Um sistema de controle não é constituído somente pelo controlador, mas também por muitos outros subsistemas que são responsáveis por condicionar, alterar e amplificar os sinais ao longo de todo o sistema.

A exemplo disso temos os principais elementos, como o atuador, o sensor, amplificadores e condicionadores de sinais em geral.

O atuador é um transdutor do sinal de controle para a planta, de modo que o controlador por si só não é conectado diretamente no processo, porém ele tem o seu sinal enviado a um atuador que traduz o sinal de modo que a planta receba a excitação necessária para se ter a saída desejada, por exemplo, em um sistema de caixa d'água onde se quer controlar o nível de água o meu atuador pode ser considerado como a bomba d'água que recebe o sinal de um controlador e injeta água na caixa d'água em uma certa quantidade para se ter um volume de água desejado, ou ainda de forma mais simplificada pode-se pensar de um atuador sendo um volante de carro, em que o motorista gira o volante até uma posição desejada esperando que o carro siga determinada direção, ou seja, o atuador é a interface entre o controlador e o sistema, ainda no caso do carro, o controlador seria o motorista, e o sistema seria o próprio carro se movendo para uma determinada posição.

O sensor é um elemento de transdução da saída, há vários tipos de sensores por exemplo, sensores de presença, movimento, rotação, nível, velocidade, temperatura, em linhas gerais a função de um sensor é justamente fornecer a um controlador a medição de uma variável de saída desejada. De modo que, no caso de muitas aplicações em eletrônica e automação, tem-se que um valor de saída é convertido para um valor de tensão, ou corrente para ser lido pelo controlador. Exemplo prático de sensores são, sensores de temperatura, os quais fornecem os

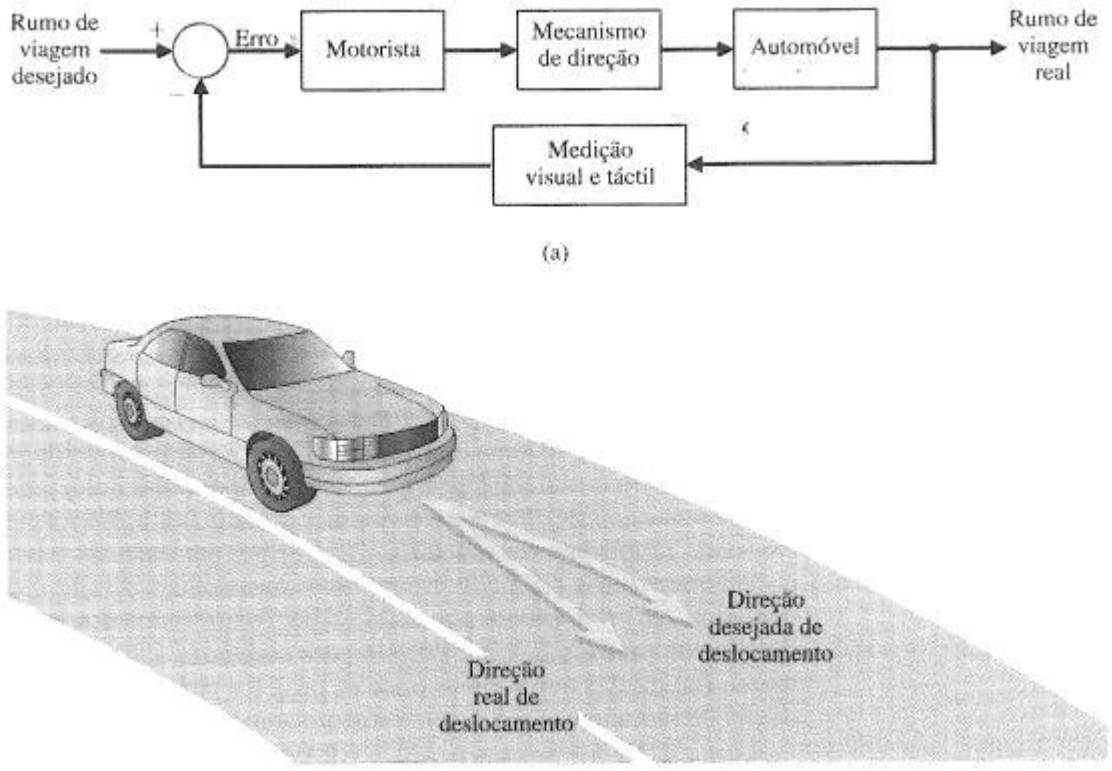
valores referentes a temperatura no instante calculado, outro exemplo também é relacionado a um sensor de rotação em que através de um instrumento o controlador tem acesso ao valor de rotação naquele exato momento.

De posse de sensores e atuadores para se ter o controle e acesso as variáveis de uma planta são necessários condicionadores e amplificadores de sinais para se fazer a interface entre esses transdutores e o controlador. Tais condicionadores de sinais tem o intuito de preparar os sinais lidos ou recebidos para serem compatíveis e nos níveis desejados para o controlador. Exemplo de condicionadores de sinais são os amplificadores operacionais que podem abaixar ou aumentar tensões proporcionalmente para a leitura ou escrita pelo controlador, também se tem os filtros digitais e analógicos que tem como intuito tratar sinais de sensores e extrair deles somente o sinal de interesse rejeitando todo tipo de sinal fora dessa faixa.

3.2.3 Exemplo de Projeto de Controle

Por fim com todos esses conceitos tem-se com um todo um sistema de controle constituído por cada uma dessas partes, a exemplo segundo (DORF e BISHOP, 2001), temos um sistema de controle de direção assistido, nos quais amplificadores hidráulicos são usados para amplificação de força aplicados aos freios ou direção e assim conseguir se ter o controle do veículo (Figura 17).

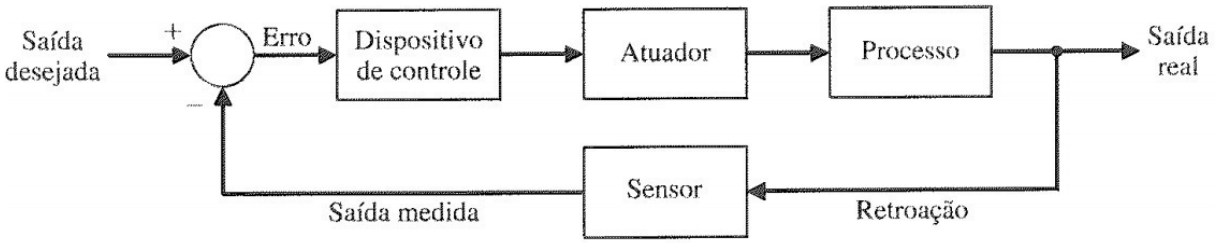
Figura 17 - Diagrama em blocos ilustrando um sistema de controle de um veículo



Fonte: (DORF e BISHOP, 2001)

A ilustração desse sistema exemplifica um controle de deslocamento em um veículo em que a referência, o objetivo é o rumo da viagem e o motorista (como controlador do veículo) manipula os atuadores disponíveis para seguir o rumo de viagem normal. O sistema em malha fechada pode ser visto de modo que o motorista (controlador) aciona os mecanismos de direção (atuadores) para acionar e mover o carro (a planta ou processo), desse modo através de uma medição visual da direção do carro e a direção da pista e táctil pelo deslocamento do volante, o rumo da viagem real é comparado com o rumo da viagem desejado para assim o motorista ter uma diferença entre o real e o projetado e a partir disso poder atuar na planta, fazendo-a seguir toda a referência desejada. O sistema pode ser comparado com um sistema de controle genérico (Figura 18).

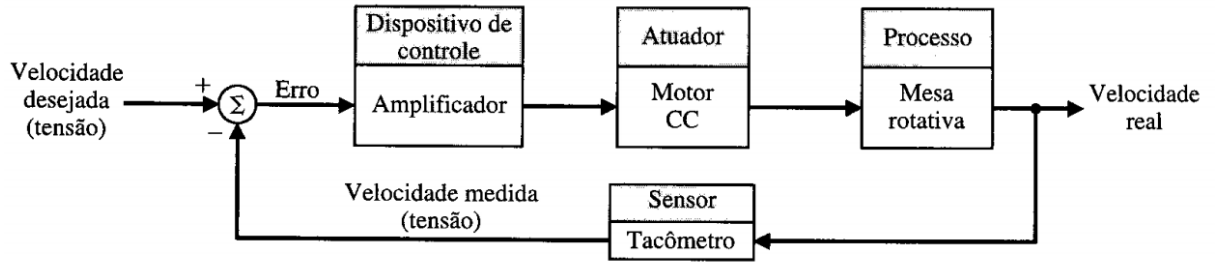
Figura 18 - Diagrama em blocos de um sistema de controle genérico



Fonte: (DORF e BISHOP, 2001)

Por fim, um exemplo de projeto de controle é o sistema de controle de velocidade de uma mesa rotativa, muito parecido com o tipo de sistema do presente trabalho. Segundo (DORF e BISHOP, 2001) muitos dispositivos usam uma mesa rotativa para girar um disco com velocidade constante, por exemplo, leitor de CD, toca disco, entre outros, todos exigem uma velocidade constante a despeito do desgaste do motor, de variação da varga e de alterações de outros componentes. Um diagrama em blocos é mostrado na Figura 19, onde se tem uma velocidade desejada com referência por um sinal de tensão, que é comparado com a velocidade real (gerado pelo tacômetro) gerando um sinal de erro, que por sua vez passa por um dispositivo de controle (em essência um amplificador) que controla o atuador do processo que no caso temos o motor CC controlando a mesa rotativa

Figura 19 - Diagrama em blocos de uma mesa rotativa



Fonte: (DORF e BISHOP, 2001)

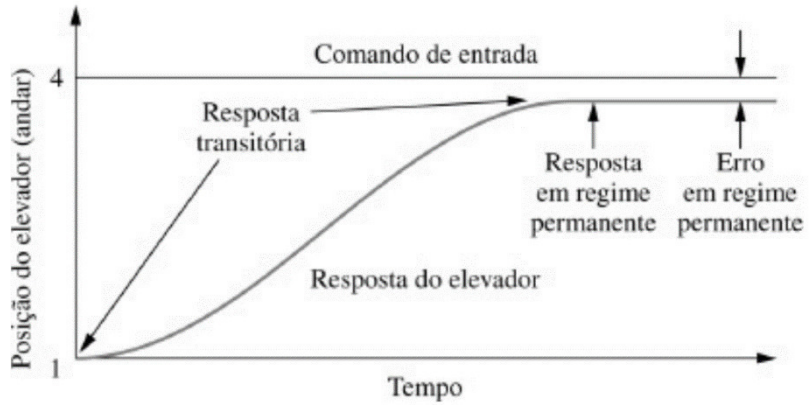
3.2.4 Estudo de Desempenho de sistemas controlados

Há alguns requisitos que se deve levar em conta em um projeto de controle, alguns deles são explicados a seguir:

> Qualquer sistema de controle deve ser estável. Além da estabilidade absoluta, um sistema de controle deve possuir uma estabilidade relativa razoável; isto é, a velocidade de resposta deve ser razoavelmente rápida e esta resposta deve possuir um amortecimento razoável. Um sistema de controle também deve ser capaz de reduzir erros a zero ou a algum valor pequeno tolerável. Qualquer sistema de controle útil deve satisfazer estes requisitos. (OGATA, 1985)

A exemplo de requisitos, especificações e estudo de respostas em um sistema, temos de acordo com (NISE, 2012) uma representação de resposta de um elevador a um comando de entrada: (Figura 20)

Figura 20 - Exemplos de resposta a um sistema de resposta à um comando em elevador



Fonte: (NISE, 2012)

Em relação ao tipo de resposta apresentado de um elevador (Figura 21) podemos explorar alguns conceitos de resposta de sistemas, por exemplo:

a) resposta transitória, geralmente vista como o tempo necessário para o sistema ir de uma posição a outra especificada, no caso do elevador esse tempo estar relacionado a velocidade em que o elevador chegará a outro andar, tal tempo dever ser pensado para não causar transtornos nos passageiros ou excessivas oscilações na hora de frear o elevador, porém também não pode ser tão lento obviamente;

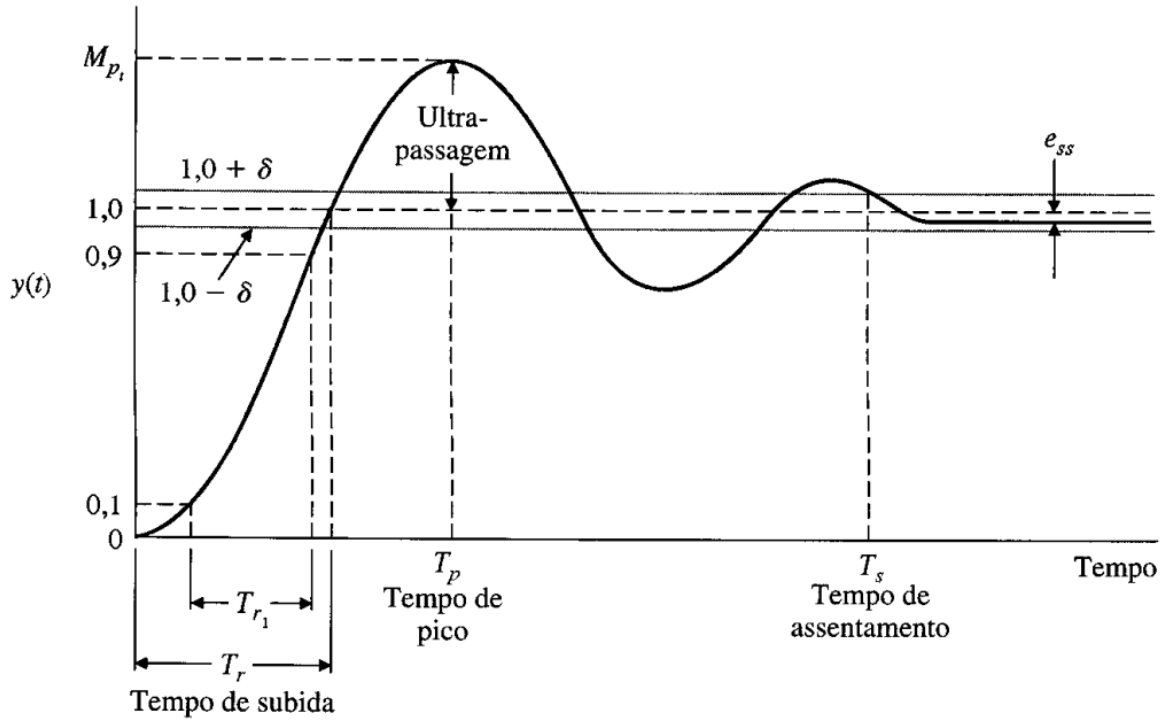
b) reposta em regime permanente, reposta pela qual o sistema se estabiliza e permanece sem relativas mudanças, muitas vezes durante o momento em que é parada a atuação do controlador, não altera mais o comando de entrada 1;

c) erro em regime permanente, também é um parâmetro muito importante relacionado ao desempenho do controlador, de modo que ele é diretamente a

diferença entre o que foi comandado e o que se tem de fato na saída. No exemplo do elevador, o erro em regime permanente deve ser obviamente levado a zero de modo que a posição do elevador seja precisamente encaixa em cada andar, muitos sistemas em geral precisam de precisão de resposta aos comandos de forma que o desempenho do controlador é muito importante.

Outros exemplos de sistemas possuem o tipo de resposta a uma entrada especificada como mostra a figura seguinte, em que se é analisado outros elementos de forma mais analítica. (DORF e BISHOP, 2001)

Figura 21- Exemplo de resposta e análise de parâmetros de um sistema



Fonte: (DORF e BISHOP, 2001)

a) tempo de subida, relacionada como citado anteriormente a rapidez com que o sistema chega ao valor de referência;

b) tempo de pico, desempenho de tempo usado em sistemas onde há um *overshoot*, ou seja, uma ultrapassagem do valor de referência;

c) ultrapassagem percentual, cálculo da porcentagem do valor ultrapassado em relação ao valor da reposta em regime permanente;

d) tempo de assentamento (ou tempo de acomodação), tempo requerido para a resposta do sistema permanecer com valor no interior de uma certa faixa percentual;
Essas curvas citadas anteriormente são relacionadas a uma entrada em degrau, ou seja, uma entrada que tem uma mudança imediata de um valor para outro de referência.

3.3 Ação de Controle

De acordo com (OGATA, 1985), temos que:

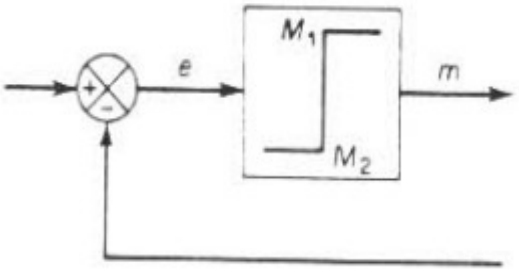
> Um controlador automático compara o valor real da saída do processo com o valor desejado, determina o desvio, e produz um sinal de controle que reduz o desvio a um valor nulo ou muito pequeno. A maneira pela qual o controlador automático produz o sinal de controle é denominada ação de controle.

A seguir será apresentado algumas ações de controle do tipo: On/off, Proporcional, Integral, Proporcional Integra (PI), Proporcional Derivativa (PD), e Proporcional Integrativa Derivativa (PID).

3.3.1 Controlador On/Off

Uma das ações de controle mais simples e intuitivas é a ação de on/off, simbolizada pela Figura 22:

Figura 22 diagrama em blocos de uma ação de controle on/off



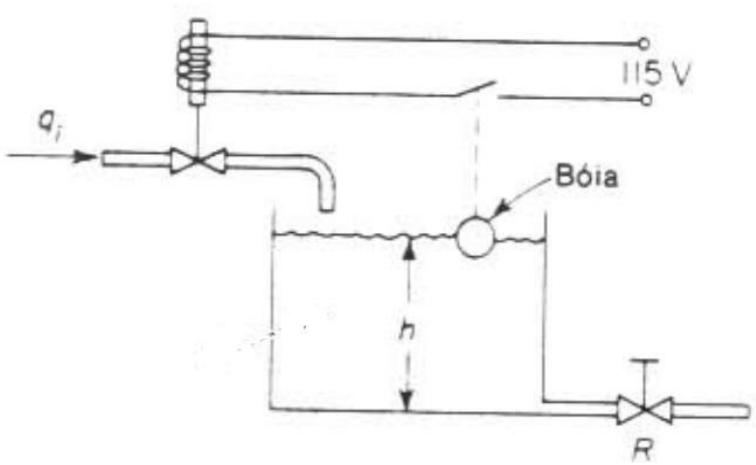
Fonte: (OGATA, 1985)

Em que o controlador só possui duas posições, ligado ou desligado, nesse tipo de controle se o erro for maior que zero, então o controle assume um valor máximo e se for menor que zero o controlador assume um valor mínimo.

Esse é um tipo de controle muito usado em caixas d'água, com um valor de histerese diferencial para prevenir a trepidação da chave, essa histerese pode ser vista como uma folga para o controle voltar a atuar, com intenção de eliminar pequenas variações no nível da água, a

Figura 23 a seguir mostra um controle simples de on/off para manter um nível de água em um reservatório. (OGATA, 1985)

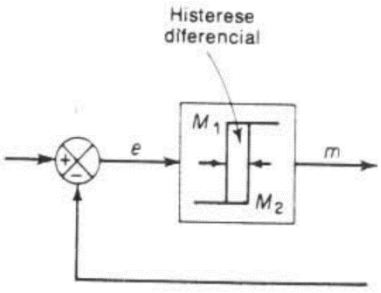
Figura 23 - Exemplo de um sistema de controle de nível de água



Fonte: (OGATA, 1985)

O sistema atua da seguinte forma, quando o nível de água do reservatório está abaixo do nível ideal, o nível h, a boia (atuando como um sensor de nível), ela fecha o circuito de energização do conjunto solenoide válvula (atuador do sistema de controle) abrindo a válvula e aumentando o fluxo de líquido para dentro do reservatório. Quando a água chega no nível máximo h a boia atua abrindo o circuito e desenergizando o solenoide e fazendo a válvula voltar para a posição de fechada interrompendo o fluxo de entrada de água no reservatório, e ela permanece nessa região até que o nível de água desça novamente. O controle de histerese como simbolizado na Figura 24, consiste em ter um faixa de tolerância para a boia voltar energizar o circuito novamente para evitar que a chave fique energizando e desenergizando a qualquer pequena variação no nível da água.

Figura 24 - Bloco diagrama representando um controle de histerese diferencial



Fonte: (OGATA, 1985)

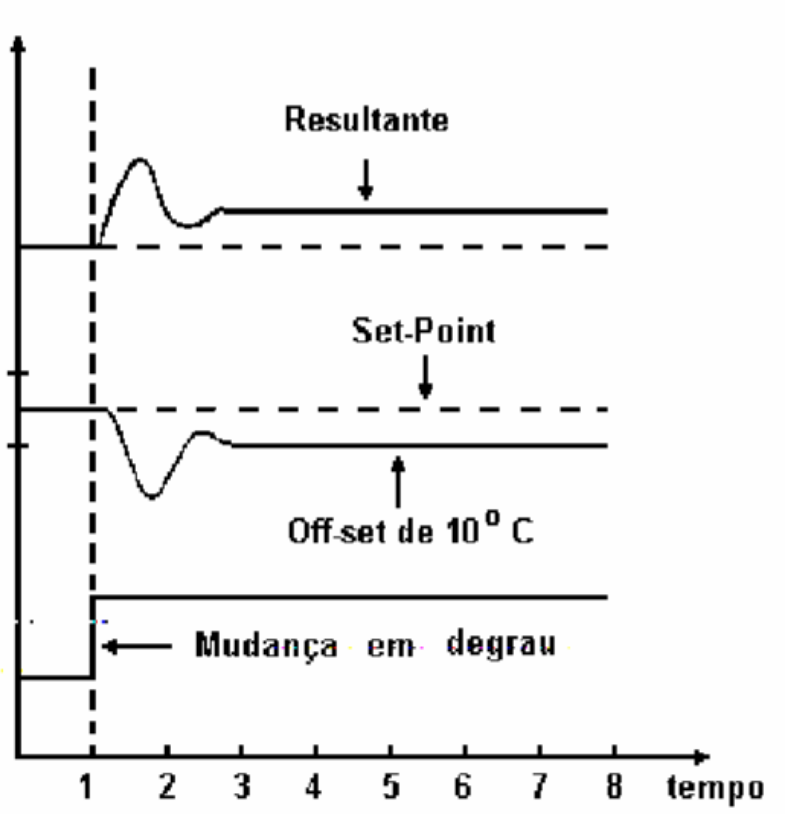
3.3.2 Ação de Controle Proporcional

De acordo com Ogata (1985), para um controlador com ação de controle proporcional, a relação entre a saída do controlador [$u(t)$] e o sinal de erro atuante[$e(t)$] é uma constante denominada K_p.

$$u(t) = Kp * e(t) \rightarrow Em\ Laplace \rightarrow \frac{U(s)}{E(s)} = Kp \qquad (1)$$

> Independente do mecanismo real, ou da forma da potência de operação o controlador proporcional é essencialmente um amplificador com um ganho ajustável.
> A característica da ação proporcional é de acelerar a resposta da variável do processo, após uma sequência de variações da própria variável ou mudança do set-point. O estudo da ação proporcional sobre um processo em malha fechada mostra que a correção da ação proporcional deixa sempre um *offset*, ou seja, não elimina totalmente o erro como mostra a Figura 25. (TEIXEIRA e PETROBRAS, 2006)

Figura 25 - características de uma ação diferencial de um sistema a uma mudança de degrau



Fonte: (TEIXEIRA e PETROBRAS, 2006)

3.3.3 Controlador Integral Puro

De acordo com Ogata (1985), em um controlador com a ação de controle integral o valor de saída do controlador *u(t)* é variado em uma taxa proporcional ao sinal erro atuante *e(t)*. Isto é:

$$\frac{d\,u(t)}{dt} = Ki * e(t) \rightarrow\ u(t) =\ Ki * \int_0^t e(t)\,dt \qquad (2)$$

$$ou\ em\ Laplace:\ sU(s) = Ki * E(s) \rightarrow \frac{U(s)}{E(s)} = Ki \qquad (3)$$

Se o valor de *e(t)* é dobrado, então o valor de *u(t)* varia duas vezes mais rápido. Para erro atuante nulo, o valor de u(t) permanece estacionário.

3.3.4 Controlador Proporcional Integral (PI)

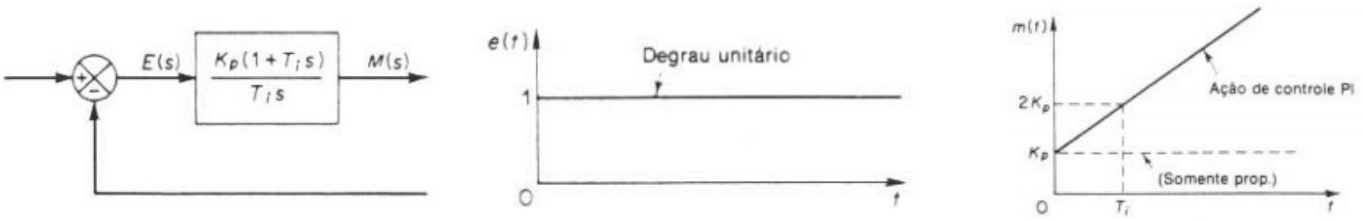
De acordo com Ogata (1985), a ação de um controle PI é:

$$u(t) = Kp * e(t) + \frac{Kp}{Ti} * \int_0^t e(t)\,dt \qquad (4)$$

Kp é o ganho e Ti representa o tempo integral. A principal característica da ação integral diminuir o erro em regime permanente à zero.

O tempo integral ajusta a ação de controle integral, enquanto uma mudança no valor de Kp afeta tanto a parte proporcional como a parte Integral da ação de controle. Como ilustrado na Figura 26.

Figura 26 - Característica da resposta de uma ação PI a um degrau unitário



Fonte: (OGATA, 1985)

3.3.5 Controlador Proporcional Derivativo (PD)

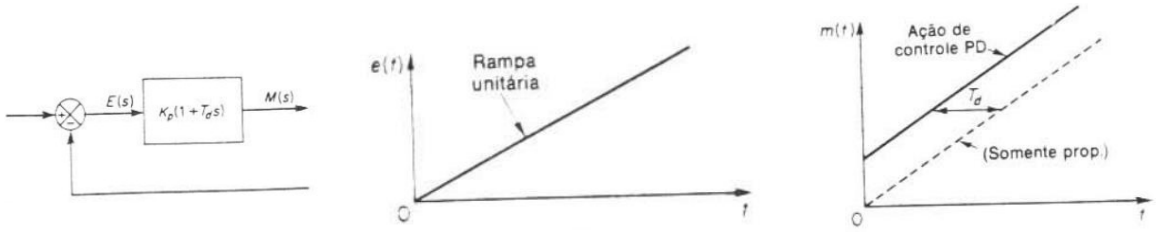
A ação de controle de um controlador proporcional derivativo é dada como:

$$u(t) = Kp * e(t) + Kp * Td * \frac{d}{dt} e(t) \qquad (5)$$

A ação de controle derivativa, algumas vezes, denominada controle de taxa, é onde a magnitude da saída do controlador é proporcional à taxa de variação do sinal erro atuante. O tempo derivativo T_d é o intervalo de tempo pelo qual a ação de taxa avança o efeito da ação de controle proporcional.

De acordo com Ogata (1985), se o sinal de erro é uma rampa unitária (Figura 27), de modo que a ação de controle tem um caráter antecipatório porém essa ação não pode antecipar uma ação que não aconteceu, exemplificando o porquê dessa ação não poder ser usada sozinha, pois ela só atua nos regimes transitórios.

Figura 27 - Resposta de uma ação PD a uma entrada em rampa unitária



Fonte: (OGATA, 1985)

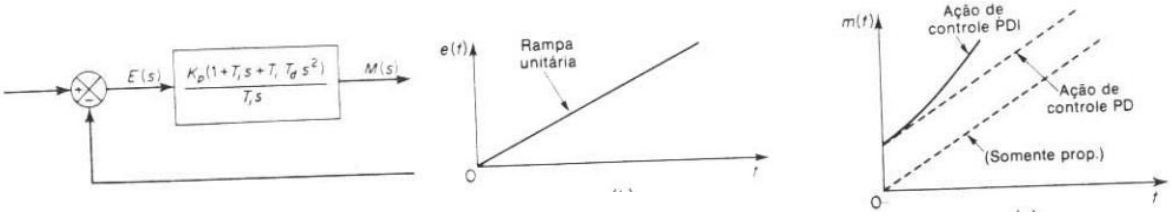
3.3.6 Controlador Proporcional Integral Derivativo (PID)

A ação de controle PID (Figura 28) combina a ação de cada controle (proporcional, integral e derivativo), de modo que assume as vantagens e características de cada uma das 3 ações, a equação do controlador é dada por:

$$u(t) = K_p * e(t) + \frac{K_p}{Ti} * \int_0^t e(t)dt + K_p T_d * \frac{d}{dt} e(t) \qquad (6)$$

$$ou\ em\ Laplace: \frac{U(s)}{E(s)} = Kp\left(1 + T_d s + \frac{1}{T_i s}\right) \qquad (7)$$

Figura 28 - Resposta a ação de controle PID a uma entrada em rampa unitária



Fonte: (OGATA, 1985)

3.4 Teoria e aspectos de sistemas de controle discreto

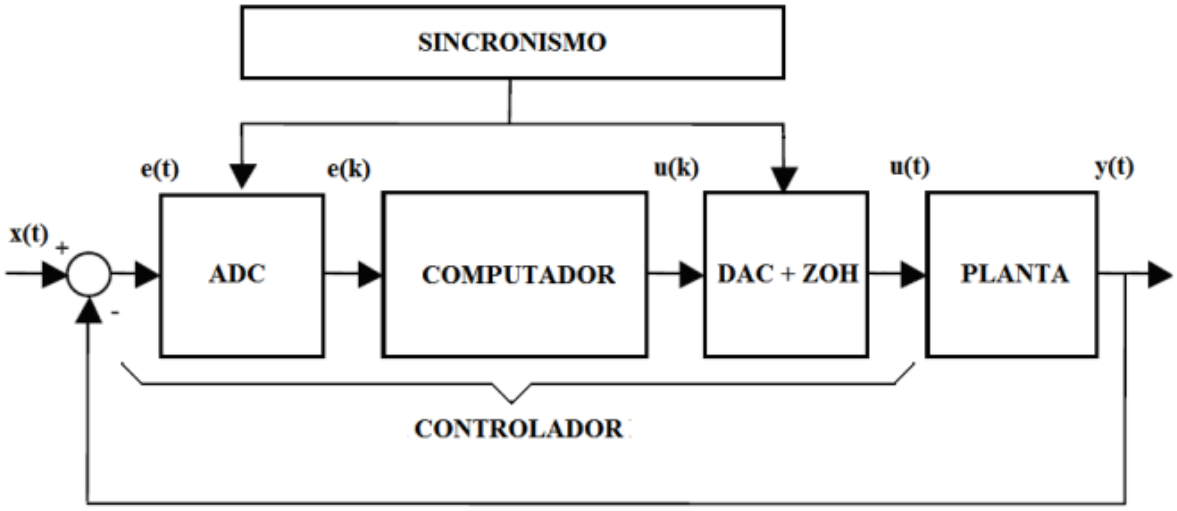
Grande parte dos controladores são implementados em circuitos digitais, fazendo - se assim necessário o projeto e conhecimento de controle em tempo discreto, ou seja o controlador não trabalha mais em tempo contínuo, dado a necessidade das conversões de analógica-digital e digital-analógica e processamento de linhas de código por tempo definido (*clock*) nos controladores digitais. Para esse projeto é necessário conhecer algumas

características básicas de sistemas de controle digital, tais como tratamento de sinais digitais, discretização de controladores etc.

3.4.1 Discretização de sistemas

Para exemplificar um sistema de controle discreto tem-se a Figura 29, onde os sinais contínuos no tempo são tidos em função (t) e os amostrados (discretos) são tidos em função de (k). de acordo com (MORENO, 2011) o comportamento de um computador no lugar de um controlador analógico pode ser descrito pela outra figura. De modo que a partir do sinal de erro e(t) acontece uma transformação do sinal para ser processado pelo controlado e por fim passa pela transformação inversa para poder controlar a planta.

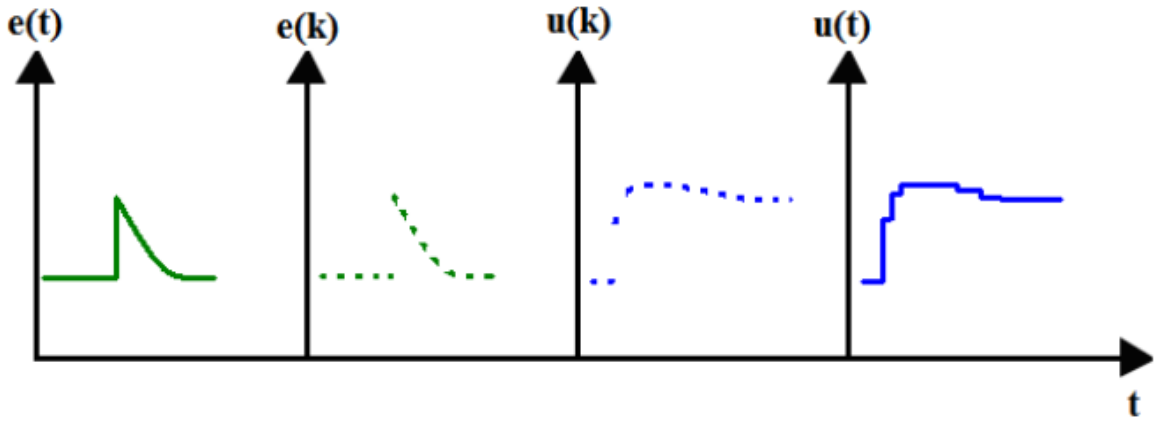
Figura 29 - Discretização de um controlador analógico



Fonte: (MORENO, 2011)

No exemplo dado a discretização do sinal (Figura 30) ocorre dentro do conversor analógico digital (ADC), depois processado pelo computador e transformado em um sinal contínuo novamente para a entrada da planta, observa-se que o sistema de controle possui sincronismo entre os elementos de conversão e cálculo do controlador, através de um sinal de sincronismo, como ilustrado no esquemático XX. Observa-se que o sinal depois de calculado é entregue a um conversor digital analógico com o intuito de converter uma palavra em bits em um sinal contínuo com o auxílio do bloco ZOH que é o segurador de ordem zero, que garante que nos intervalos entre duas amostras será mantido o valor da amostra anterior,

Figura 30 - Sinais discretizados e contínuos do sistema da Figura 29



Fonte: (MORENO, 2011)

3.4.2 Métodos de Discretização de Sistemas

De acordo com Moreno (2011), os métodos seguintes são usados para aproximação do termo integral em sistemas discretos, podendo para isso ser utilizado triângulos ou trapézios como será mostrado a seguir.

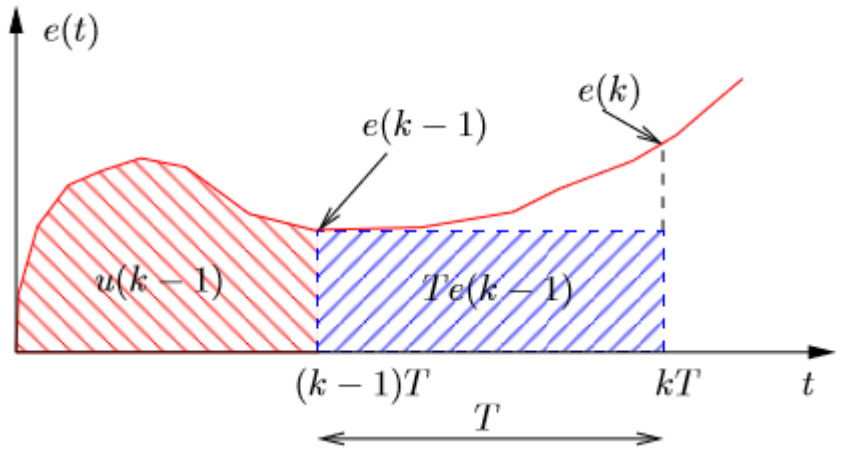
3.4.2.1 Método da Diferença Adianta (Forward Difference)

Conforme Lages (2010) apud Moreno (2011), nesta aproximação considera-se que o valor de e(t), no intervalo $[(k-1)T\,, kT)]$, é constante e dado por:

$$e\frac{(k-1)}{T} = e(k-1) \qquad (8)$$

, de modo que isto implica em tomar a área de uma retângulo como aproximação para a integral como mostra a Figura 31:

Figura 31 - Cálculo da integral pelo método da diferença adiantada



Fonte: (LAGES, 2011)

Logo, a integral de e(t) nos intervalos de discretização pelo método apresentado será de

$$Área = Integral = T * e[k-1]$$

De modo que o termo integral ao longo do tempo fica:

$$u[k] = u[k-1] + T * e[k-1]$$

Aplicando a transformada z a essa expressão temos:

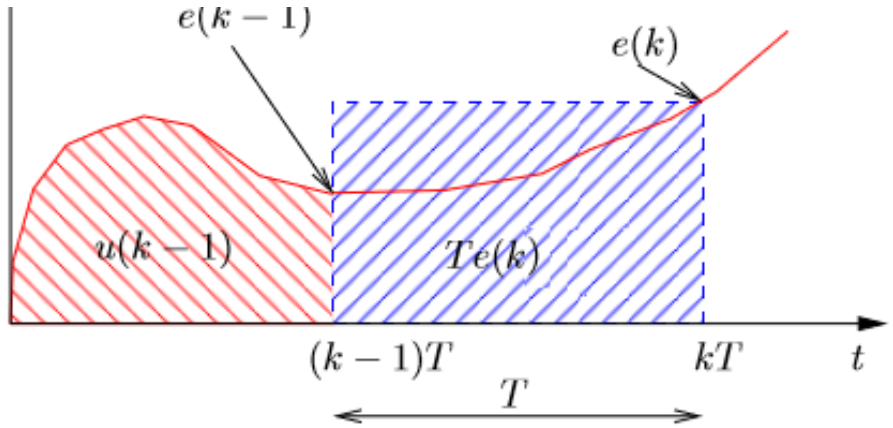
$$U[z] = z^{-1}u[z] + Tz^{-1}E[z] \rightarrow \frac{U[z]}{E[z]} = H[z] = \frac{T}{z-1} \cong \frac{1}{s} \qquad (10)$$

3.4.2.2 Método da diferença atrasada (Backward Difference)

Conforme Lages (2010) apud Moreno (2011), nesta aproximação considera-se que o valor de e(t), no intervalo $[(k-1)T\,, kT)]$, é constante e dado por:

$e(kT) = e(k)$, de modo que isto implica em tomar a área de uma retângulo como aproximação para a integral como mostra a Figura 32:

Figura 32 - Cálculo da Integral pelo método da diferença avançada



Fonte: (LAGES, 2011)

Logo a integral de e(t) nos intervalos de discretização pelo método apresentado será de:

$$Área = Integral = T * e[k]$$

De modo que o termo integral ao longo do tempo fica:

$$u[k] = u[k-1] + T * e[k]$$

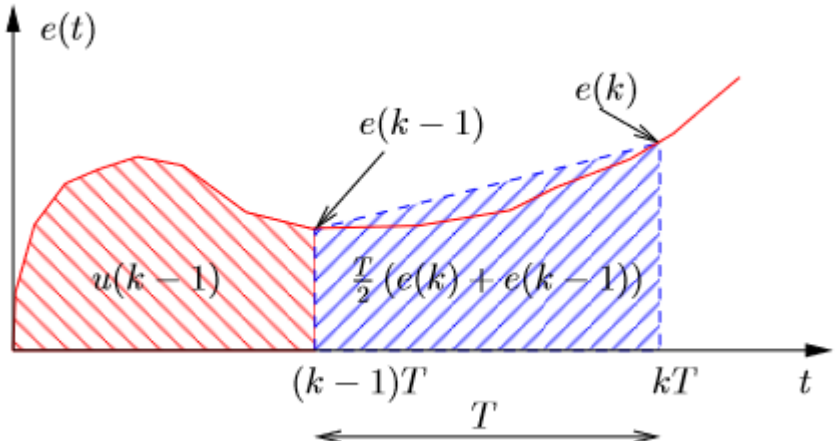
Aplicando a transformada z a essa expressão temos:

$$U[z] = z^{-1}u[z] + TE[z] \rightarrow \frac{U[z]}{E[z]} = H[z] = \frac{Tz}{z-1} \cong \frac{1}{s} \qquad (11)$$

3.4.2.3 Transformação Bilinear (Aproximação de Tustin)

Conforme Lages (2010) apud MORENO (2011), nesta aproximação considera-se que o valor de e(t), no intervalo $[(k-1)T\,, kT)]$, é constante e dado pela média entre $e[(k-1)T] = e(k-1)$ e $e(kT) = e(k)$, de modo que isto implica em tomar a área de uma retângulo como aproximação para a integral como mostra a Figura 33:

Figura 33 - aproximação da integral pelo método de Tustin



Fonte: (LAGES, 2011)

Logo a integral de e(t) nos intervalos de discretização pelo método apresentado será de:

$$Área = Integral = \frac{T}{2}(e[k-1] + e[k])$$

De modo que o termo integral ao longo do tempo fica:

$$u[k] = u[k-1] + \frac{T}{2}(e[k] + e[k-1])$$

Aplicando a transformada z a essa expressão temos:

$$U[z] = z^{-1}u[z] + \frac{T}{2}(E[z] + z^{-1}E[z]) \rightarrow \frac{U[z]}{E[z]} = H[z] = \frac{T}{2}\frac{(z+1)}{(z-1)} \cong \frac{1}{s} \qquad (12)$$

3.4.3 Controlador PID Digital

Como já estudado a função de transferência de um controlador PID contínuo é escrita como:

$$\frac{U(s)}{E(s)} = Kp + \frac{Ki}{s} + sK_d \qquad (13)$$

De acordo com (MORENO, 2011) para se obter a versão discreta deste controlador poderia se utiliza qualquer um dos métodos apresentado na seção anterior, de modo a simplificar o projeto do controlador foi-se escolhido o método de (*backward difference)*.

Portanto, fazendo a seguinte consideração de aproximação

$$s = \frac{z-1}{zT}$$

Temos:

$$\frac{U(z)}{e(z)} = K_p + K_i\frac{(z\,T)}{z-1} + K_d\frac{(z-1)}{zT}$$

Reagrupando os termos:

$$u(z)(T - Tz^{-1}) = e(z)[z^2(K_pT + K_iT^2 + K_d) - z(K_pT + 2K_d) + K_d)]$$

Dividindo-se a equação por z², tem-se:

$$u(z)(T - Tz^{-1}) = e(z)(K_pT + K_iT^2 + K_d - z^{-1}(K_pT + 2K_d) + z^{-2}K_d)$$

Fazendo-se uso da identidade:

$$z^k u[k] = u[n + k]$$

Tem-se finalmente:

$$u[n] = u[n-1] + e[n]\left(K_p + K_iT + \frac{K_d}{T}\right) - e[n-1]\left(K_p + \frac{2K_d}{T}\right) + e[n-2]\frac{K_d}{T} \qquad (14)$$

Ou em função dos Ganhos:

$$u[n] = u[n-1] + K_p(e[n] - e[n-1]) + K_iT(e[n])$$

$$+ \frac{K_d}{T}(e[n] - 2e[n-1] + e[n-2]) \qquad (15)$$

De modo que:

$$K_i = \frac{K_p}{T_i} \; e \; K_d = K_pT_d$$

Kp, Ki, Kd são respectivamente o ganho proporcional, Integral e derivativo e Ti, Td, T são respectivamente o tempo integral, derivativo e a taxa de amostragem.

4. CARACTERIZAÇÃO DA PLANTA

4.1 MOTOR DC E TACO GERADOR

O motor elétrico de corrente contínua é uma máquina de corrente contínua que transforma energia elétrica (com tensão contínua) em energia mecânica no eixo do motor.

A relação básica entre a conversão de energia eletromecânica é de que a energia elétrica entregue ao motor é convertida em energia mecânica, considerando que a perdas no sistema, a energia elétrica nem sempre é convertida completamente em energia mecânica, daí vem as medidas de rendimento do motor, comparando a potência na entrada do sistema motor e a potência de saída real.

As grandezas de potência elétrica são tensão e corrente:

$$P_{Elétrica} = V * I \quad (16)$$

Já as grandezas de potência mecânica são torque (conjugado) e velocidade:

$$P_{Mecânica} = \tau * \omega \quad (17)$$

De modo que em um motor a Potência Elétrica é convertida em Potência mecânica, ou seja, de forma simplificada:

$$P_{Elétrica} = P_{Mecânica} = V * I = \tau * \omega \quad (18)$$

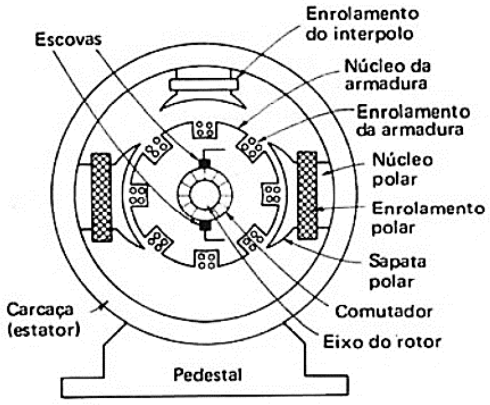
4.1.1 Aspectos construtivos do Motor DC

O motor de corrente contínua tem as duas principais estruturas que são:

a) o rotor, parte girante da máquina em que se tem o enrolamento de armadura, responsável pelas correntes e tensões de armadura e alimentação do motor;
b) o estator, parte física da máquina responsável por gerar o campo magnético que atravessa o rotor, tal campo magnético pode ser constituído pelo enrolamento de campo ou por imã permanente.

A Figura 34 mostra as principais estruturas de um motor de corrente contínua.

Figura 34 - Partes estruturais de um motor de corrente contínua



Fonte: (TORO)

Na Figura 34, mostra as escovas, que são constituídas de material condutor e responsáveis por manter o contato entre o comutador (parte girante da máquina acoplada ao enrolamento de armadura) e os condutores de alimentação em corrente contínua da armadura. O comutador, geralmente em formato de anel, é a parte condutora girante que é conectada ao enrolamento de armadura do rotor, em corrente contínua o comutador é geralmente seccionado de acordo com o número de polos da máquina para sempre ser recebida tensão e corrente contínua no anel comutador.

Na parte do estator se tem o enrolamento polar ou de campo, e o núcleo ferromagnético para condução mais apropriada do fluxo magnético gerado pelo enrolamento de campo, na continuação do núcleo polar se tem a sapata polar também constituída do mesmo material com o intuito de distribuir mais uniformemente o fluxo magnético na superfície do rotor.

Na Figura 34 também apresenta o enrolamento de interpolo, que é um enrolamento ligado em série com a armadura para corrigir alguns problemas durante a comutação gerada pelo fluxo magnético gerado pela corrente de armadura.

4.1.2 Equacionamento e Circuito Equivalente do Motor DC

As principais leis eletromagnéticas envolvidas no equacionamento de um motor são:

Lei de Faraday:

De acordo com Chapman (2013), a Lei de *faraday* afirma que, se houver um fluxo passando através de uma espera de fio condutor, então uma tensão será induzida sendo diretamente proporcional à taxa de variação do fluxo em relação ao tempo.

$$e_{ind} = -\frac{d\phi}{dt} \rightarrow ou\ \nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial \mathrm{t}} \qquad (19)$$

$$ou\ também\ pode\ ser\ escrita\ como:$$

$$e = (\boldsymbol{v} \times \boldsymbol{B}) \cdot \boldsymbol{l} \qquad (20)$$

Lei de Lorentz:

De acordo com (CHAPMAN, 2013), um segundo efeito importante de um campo magnético no seu entorno é que ele induz uma força em um fio que esteja conduzindo uma corrente dentro do campo.

$$\boldsymbol{F_{mag}} = Q\boldsymbol{v} \times \boldsymbol{B} \rightarrow \boldsymbol{F_{mag}} = \int I(d\boldsymbol{l} \times \boldsymbol{B}) \qquad (21)$$

A partir dessas equações temos as relações de torque e velocidade em uma máquina cc. De modo que de acordo com (CHAPMAN, 2013):

$$e_{ind} = \left(\frac{2}{\pi}\right) A_p B \omega_m, ou\ simplemente\ E = K\phi w_n \qquad (22)$$

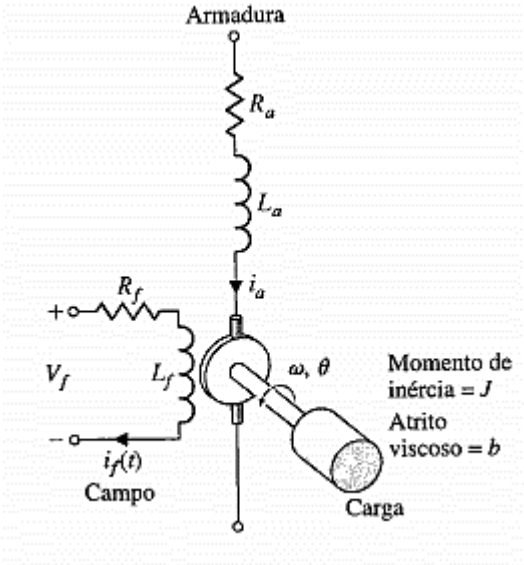
$$\tau = 2rilB, ou\ simplesmente\ \mathrm{T} = K\phi I \qquad (23)$$

Onde o conjugado é tido em função de uma constante K dependente dos parâmetros construtivos do motor, o fluxo magnético Φ em que o rotor estar inserido e a corrente I é referente a corrente de armadura que circula no enrolamento de armadura do rotor.

De modo que com base nas equações básicas de uma máquina CC, para esse presente trabalho, varia-se a tensão no motor para se obter velocidade de rotação do mesmo.

Através dessas equações e técnicas de modelagem matemáticas (DORF e BISHOP, 2001) chega as seguintes equações através do modelo equivalente mostrado na Figura 35.

Figura 35 - Modelo equivalente de um circuito de motor DC



Fonte: (DORF e BISHOP, 2001)

$$Corrente\ de\ Armadura\ Constante:$$

$$\frac{\theta(s)}{V_f(s)} = \frac{K_m}{s(Js+b)\left(L_f s + R_f\right)} \tag{24}$$

$$Corrente\ de\ Campo\ Constante:$$

$$\frac{\theta(s)}{V_a(s)} = \frac{K_m}{s[(R_a + L_a s(J_s + b)K_b K_m]} \tag{25}$$

Como a maior parte das variáveis do motor utilizado no projeto não são conhecidas e como também o foco do presente trabalho não é referente a aplicações mais complexas de controle, tomou-se como conveniente fazer o controlador de forma empírica considerando o motor como uma planta e aplicando diferente coeficientes no controlador PID e observando as respostas.

Buscou-se um controle de velocidade por variação de tensão média, através da modulação PWM, nos terminais do motor e a utilização do tacogerador para se obter os valores de velocidade do motor, de modo que através dessas duas ferramentas se tem tanto acesso tanto a entrada quanto a saída da planta para o controle de velocidade.

4.1.3 Tacogerador

O princípio de funcionamento de um tacogerador é o mesmo de um gerador em que se tem um eixo que rotaciona por energia mecânica e através disso é convertido em uma tensão e corrente no formato senoidal proporcional a frequência de rotação e número de polos, atuando desse modo como um transdutor de velocidade.

O tacogerador possui um campo formado por imã permanente de modo que na espira temos a relação de: (ANDRADE, 2008)

$\phi = \phi_0 \cos(\omega t)$, *como a tensão induziada é dada por*

$$e = -\frac{d\omega}{dt} \ \rightarrow \ e = \phi_0 \omega \sin(\omega t) \tag{26}$$

Temos que a tensão induzida no tacogerador será relacionada com a tensão na espira pela relação:

$$E_{taco} = K_t * e \tag{27}$$

Em que K_t é chamada constante taquimétrica.

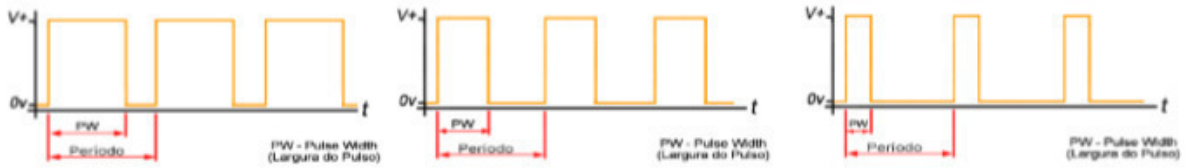
4.1.4 Aplicações e Características do controle por PWM

A modulação por Largura de Pulso ou PWM (*Pulse Width Modulation)* é muito usada em controle de potência onde se deseja uma tensão média variável, ou em controle de tensão em motores, controle de chaves de potência de fontes chaveadas e conversores de vários, tipos ou em até controle de servos motores entre outras aplicações.

Essa modulação permite a separação de circuitos digitais de circuitos de potência de modo que através dela se tem o controle de chaves que ora estão abertas ora fechadas permitindo que a chave seja controlada por um sinal de liga e desliga que não consuma potência, como por exemplo um interruptor em que o operador liga e desliga o circuito sem precisar fornecer energia elétrica a ele, da mesma forma o sistema digital liga e desliga a chave sem precisar fornecer energia elétrica considerável para tal controle.

O princípio básico de um PWM é de uma onda "quadrada" (Figura 36) em que se tem uma frequência constante com um tempo em que em "on" variável, ou seja, uma largura de pulso variável. A partir dessa técnica de modulação, se tem que para uma largura de pulso de 0% uma tensão igual a 0V, e para uma largura de pulso de 100% se tem uma tensão de +VCC.

Figura 36 - Especificações de características uma onda PWM



Fonte: Adaptado

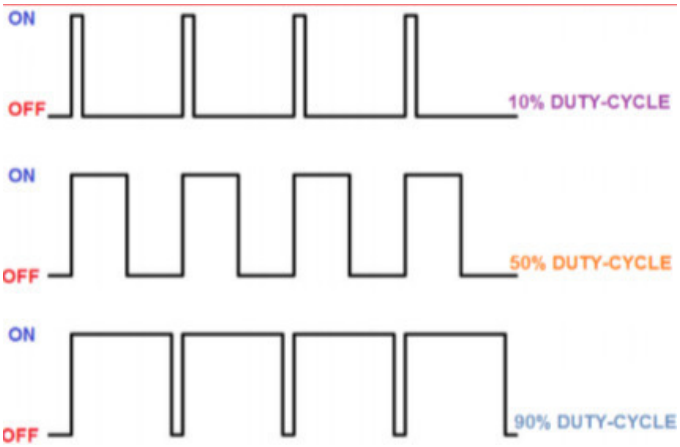
A partir disso temos como ilustrado na figura seguinte que o tempo em alta (+Vcc) somado com o tempo em baixa (0V) terá sempre

A frequência do PWM é escolhida de acordo com o tipo de aplicação e carga controlada atentando para as constantes de tempo das cargas inseridas, de modo que por exemplo para cargas de motores, que atrasos mecânicos de resposta mais lenta que a de circuito eletrônicos pode ser usada uma frequência mais baixa de PWM.

Independente da frequência a ser utilizada a técnica de PWM permite uma variação linear de tensão média a partir da variação do ciclo de trabalho como mostrado na Figura 37 a seguir em que se tem a relação de:

$$Tensão_{média} = Tensão_{Aplicada} * Duty\ Cycle \rightarrow V_{med} = Vcc * D \qquad (28)$$

Figura 37 - Onda PWM para vários valores de *Duty Cycle*



Fonte: Adaptado

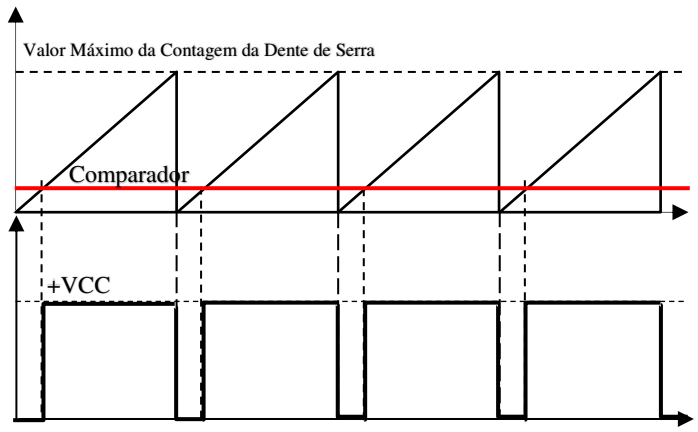
No presente trabalho o PWM é usado para controle de tensão média aplicado ao motor em que o sinal PWM entra em dois estágios de amplificação e alimenta o motor.

4.1.5 Lógica de Implementação em programação HDL

Há várias formas de se obter um sinal PWM em um FPGA, uma delas é através da comparação entre uma onda dente de serra com um comparador, de modo que frequência da dente de serra define a frequência do sinal PWM e se varia apenas o valor do comparador para se obter as diferentes larguras de pulso.

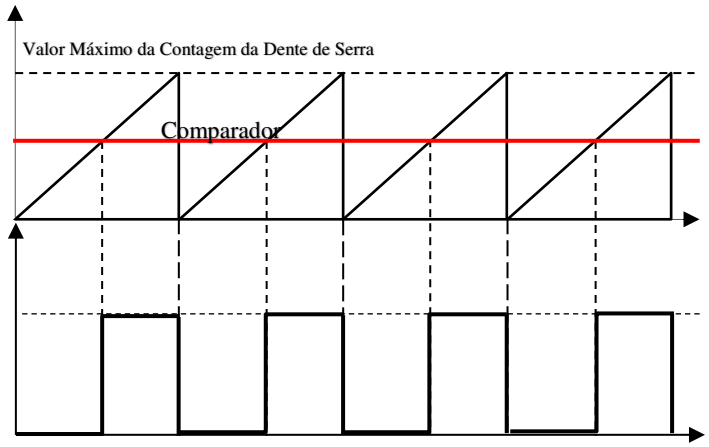
Nas figuras seguintes se tem as formas de onda para dois valores de comparadores diferentes: (Figura 38 e 39)

Figura 38 - Forma de Onda formada por dente de serra para valor 1 do comparador



Fonte: Próprio Autor

Figura 39 - Forma de Onda formada por dente de serra para valor 2 do comparador

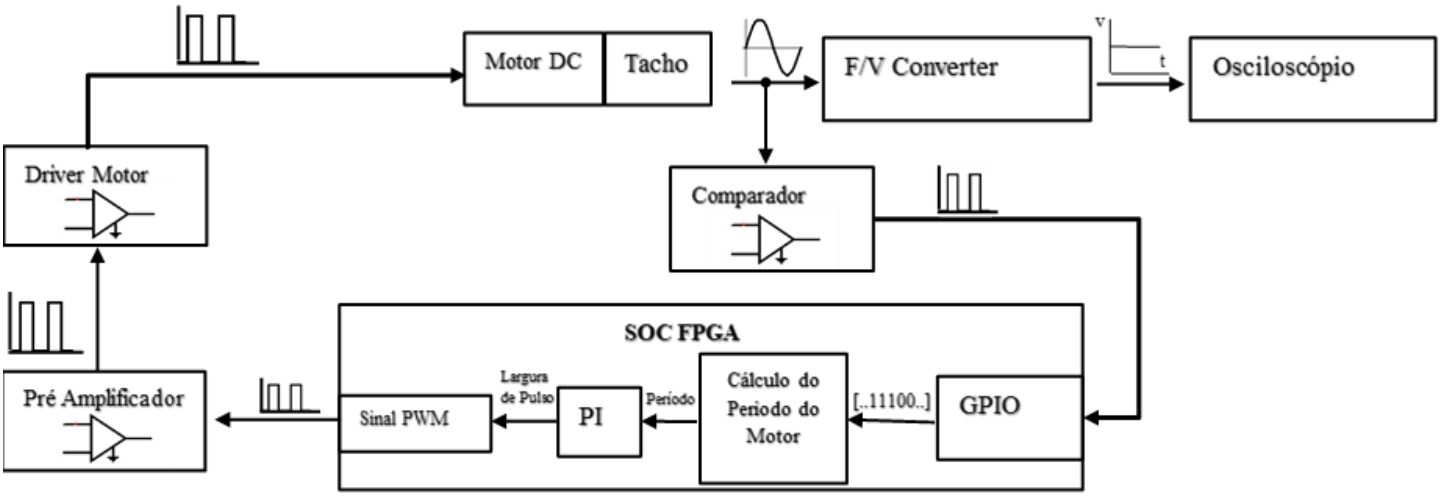


Fonte: Próprio Autor

5. IMPLEMENTAÇÃO

Para explicar o sistema de controle dividiu-se de em duas partes, a parte de entrada e de saída da planta, sendo explicado cada detalhe construtivo e de condicionamento de sinais, por conseguinte é explicado os algoritmos de sensores, atuadores, hardware e software. Segue o diagrama esquemático do sistema de controle PI (Figura 40).

Figura 40 - Diagrama esquemático do sistema de controle PI implementado



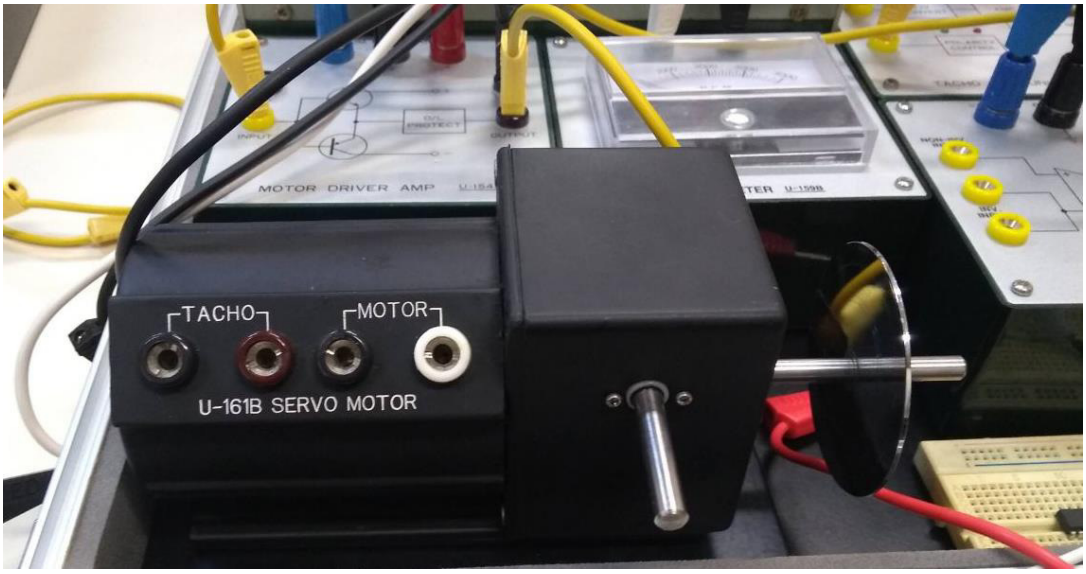
Fonte: Próprio Autor

5.1 Componentes

5.1.1 Motor DC / Tacogerador

O Motor DC (Figura 41) usado no sistema de controle é o Motor DC de 4,5W/12V do módulo de treinamento em sistemas de servo-mecanismo da Inimipa® ED – 4400B, O conjunto de motor DC e Tacogerador é especificado como U-161 pelo fabricante.

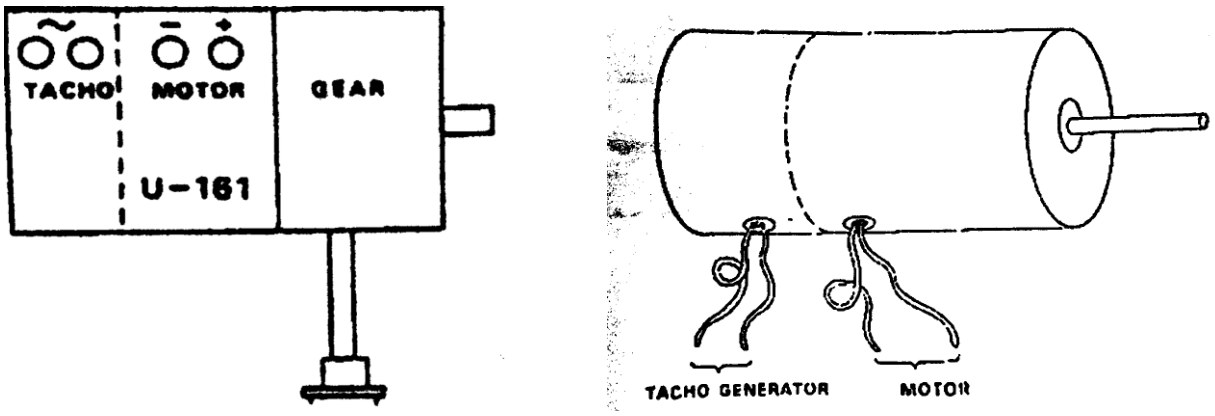
Figura 41 - Foto do Motor DC módulo 161 usado na implementação de controle



Fonte: Próprio Autor

O tacogerador usado no sistema de controle é cerca de 3V/5000 rpm, de modo que é acoplado diretamente no servo mecanismo e produz um sinal AC diretamente proporcional a velocidade de rotação do motor. A Ilustração do conjunto motor tacho o gerador é representado a seguir. (Figura 42)

Figura 42 - Ilustração do conjunto motor tacogerador

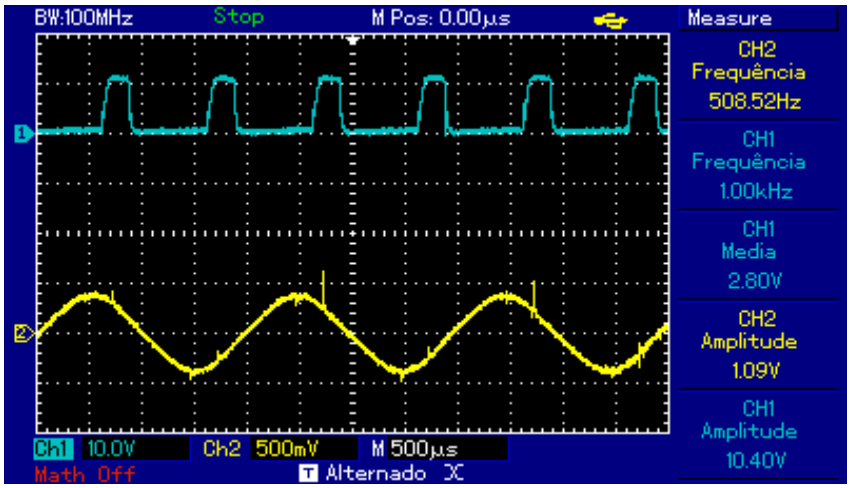


Fonte: (INIMIPA, 2004)

As figuras seguintes mostram as imagens do osciloscópio observando os valores do tacogerador para diferentes valores de velocidade do Motor. De modo que a entrada é identificada como uma onda quadrada amplificada diretamente do De1-Soc FPGA para excitação do Motor e a senoide tem o valor de tensão e de frequência proporcional a velocidade do motor.

Na Figura 43 observa-se que para um valor de modulação PWM com Duty-cycle de 24% e frequência de 1Khz, o sinal de entrada fica com a tensão média em torno de 2,8V,gerando, assim, no caso especificado a rotação do motor é em torno de 1000 rpm, com um sinal de tacogerador com tensão de amplitude de 1,09 volts, e uma frequência aproximada de 508Hz, proporcionais a velocidade do motor.

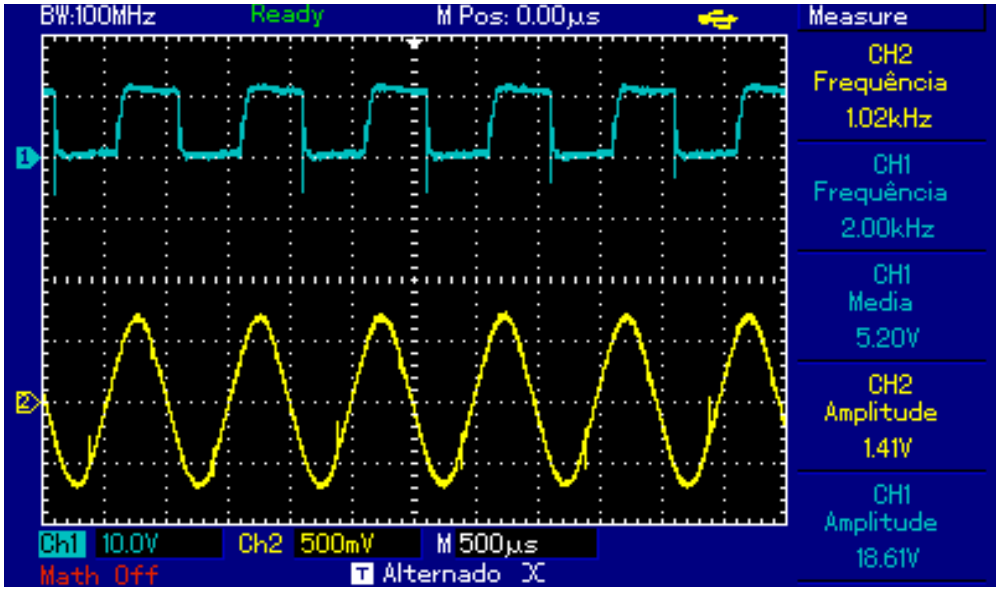
Figura 43 - Formas de onda da entrada de onda do motor (onda de cima) e saída do tacogerador (onda de baixo) para uma rotação de aproximadamente 1000rpm



Fonte: Próprio Autor

Na Figura 44 observa-se que para um valor de modulação PWM com Duty-cycle de 48% e frequência de 1Khz, o sinal de entrada fica com a tensão média em torno de 5,2V,gerando, assim, no caso especificado a rotação do motor é em torno de 2000 rpm, com um sinal de tacogerador com tensão de amplitude de 1,41 volts, e uma frequência aproximada de 1,02KHz, proporcionais a velocidade do motor.

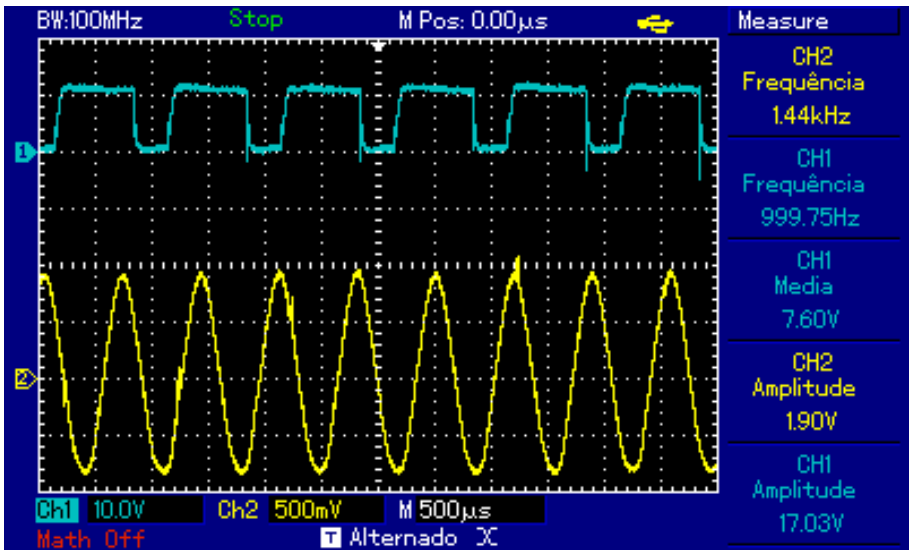
Figura 44 - Formas de onda da entrada de onda do motor (onda de cima) e saída do tacogerador (onda de baixo) para uma rotação de aproximadamente 2000rpm



Fonte: Próprio Autor

Na Figura 45 observa-se que para um valor de modulação PWM com Duty-cycle de 72% e frequência de 1Khz, o sinal de entrada fica com a tensão média em torno de 7,6V,gerando, assim, no caso especificado a rotação do motor é em torno de 3000 rpm, com um sinal de tacogerador com tensão de amplitude de 1,9 volts, e uma frequência aproximada de 1,44KHz, proporcionais a velocidade do motor.

Figura 45 - Formas de onda da entrada de onda do motor (onda de cima) e saída do tacogerador (onda de baixo) para uma rotação de aproximadamente 3000rpm



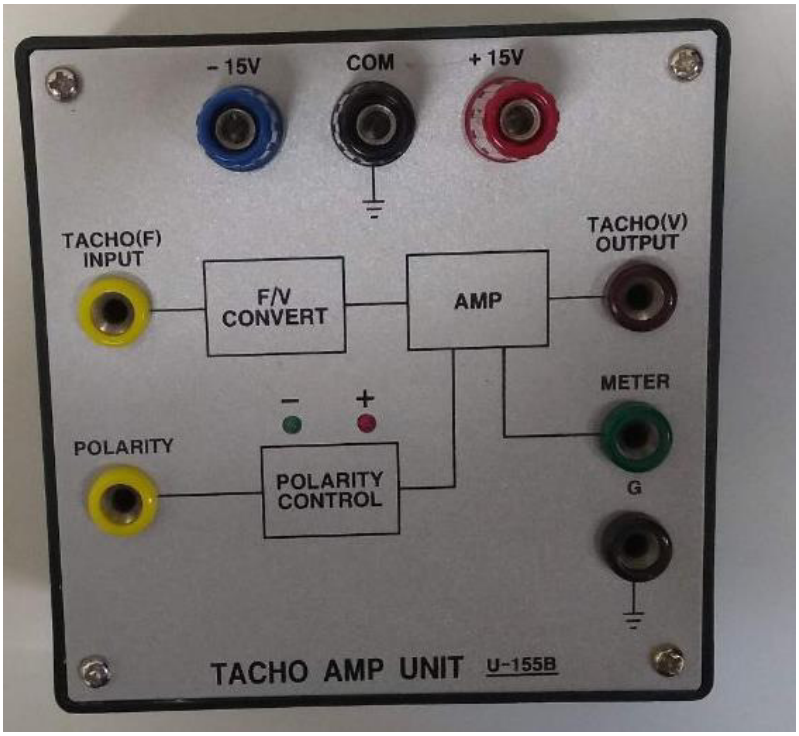
Fonte: Próprio Autor

De acordo com as formas de ondas experimentais comprava-se a proporcionalidade entre a velocidade do motor com a frequência e tensão do sinal do tacogerador. Observando que a partir do aumento da tensão média na entrada do motor, a saída de taco gerador com forma de onda senoidal tinha sua amplitude e frequência aumentada.

5.1.2 F/V Converter

Para relacionar o valor de rotação do Motor e o valor de rotação do taco gerador, usou-se o módulo *Tacho Amp Unit* U-155B (Figura 46) em que o F/V *Converter* possui a relação de 1000rpm/V.

Figura 46 - Foto do módulo Tacho Amp Unit u155B, utilizado no presente projeto



Fonte: Próprio Autor

Fazendo as devidas transformações e aproximações constatou a relação entre a frequência do Taco em (HZ) com o Motor em (Rpm). Por média simples dos valores (Tabela 2) e pensando nas grandes oscilações de velocidade do motor viu-se como conveniente deduzir experimentalmente que a o tacogerador gira a uma frequência de 30x mais rápido que o motor, ou seja, uma relação de que o valor em RPM do motor é o dobro do valor em Hz do Tacogerador.

Tabela 2 - Algum dos valores usados para a dedução da relação entre a velocidade do motor e a do tacogerador

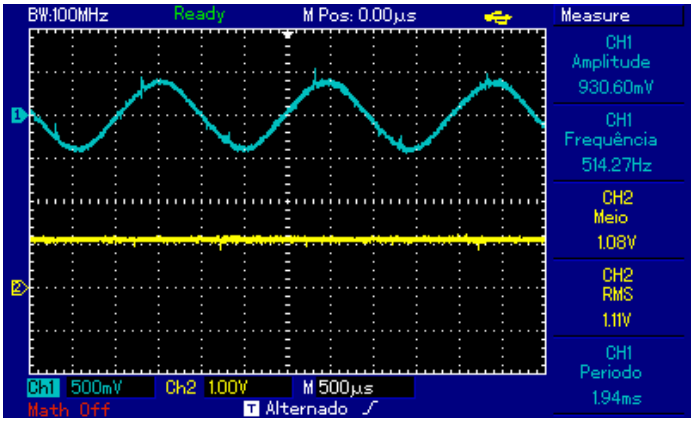
Cálculo Experimental Médio para Relação entre a Velocidade do Motor em rpm e do Sinal de Tacho em Hz			
Saída do F/V Converter (V)	Velocidade em RPM	Tacho Gerador (Hz)	Relação Motor/Tacho (RPM/Hz)
0,853	853	535,22	1,59
1,36	1360	682,17	1,99
1,95	1950	834,67	2,34
2,14	2140	929,02	2,30
2,3	2300	1028,05	2,24
2,51	2510	1104,05	2,27
2,66	2660	1236,40	2,15
3,34	3340	1453,83	2,30
3,87	3870	1686,91	2,29

Fonte: Próprio Autor

A Figura 47 mostra as formas de onda da senoide vinda do tacogerador e a segunda forma de onda é a tensão CC resultante do módulo de conversão (F/V Converter), em que se tem um valor de tensão proporcional à frequência do tacogerador, como a conversão de 1V/1000rpm.

Os valores obtidos na Figura 47 correspondem a uma velocidade de aproximadamente 1000 rpm do Motor, que é resultante de uma frequência de aproximadamente 514 Hz do tacogerador, conforme discutido anteriormente.

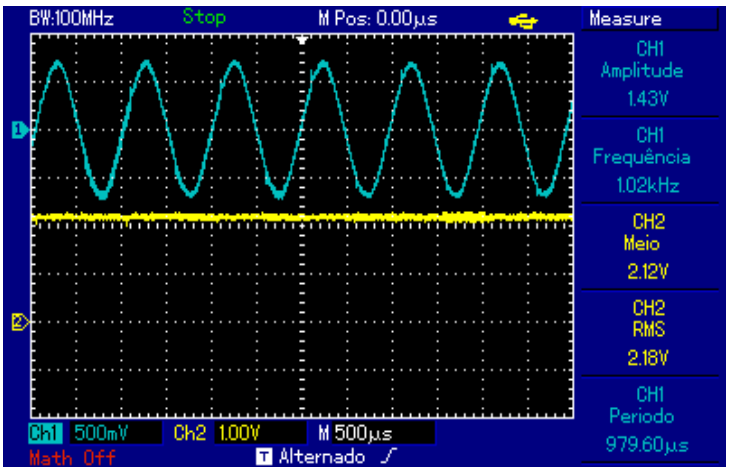
Figura 47 - Tensão resultante (onda de baixo) do módulo *F/V converter* pra a frequência do taco gerador (onda de cima) para uma velocidade de aproximadamente 1000rpm



Fonte: Próprio Autor

Os valores obtidos na Figura 48 correspondem a uma velocidade de aproximadamente 2000 rpm do Motor, que é resultante de uma frequência de aproximadamente 514 Hz do tacogerador.

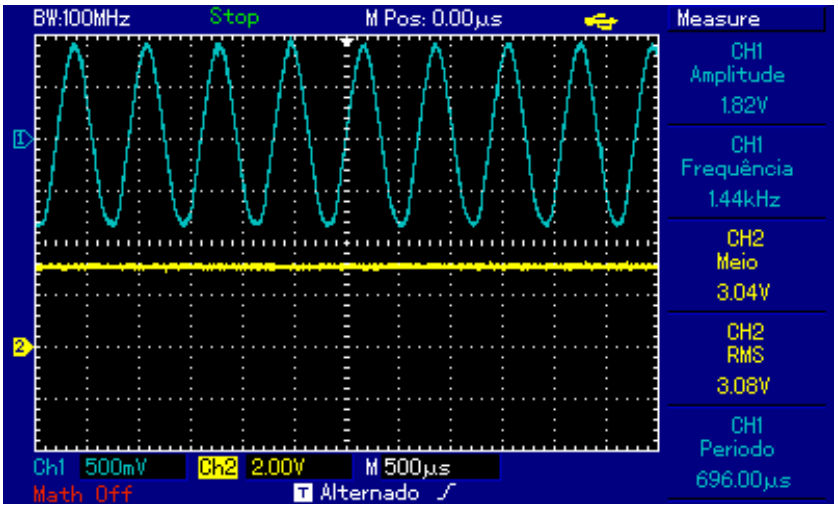
Figura 48 - Tensão resultante (onda de baixo) do módulo F/V converter pra a frequência do taco gerador (onda de cima) para uma velocidade de aproximadamente 2000rpm



Fonte: Próprio Autor

Os valores obtidos na Figura 49 correspondem a uma velocidade de aproximadamente 3000 rpm do Motor, que é resultante de uma frequência de aproximadamente 1,44KHz do tacogerador.

Figura 49 - Tensão resultante (onda de baixo) do módulo F/V converter pra a frequência do taco gerador (onda de cima) para uma velocidade de aproximadamente 3000rpm



Fonte: Próprio Autor

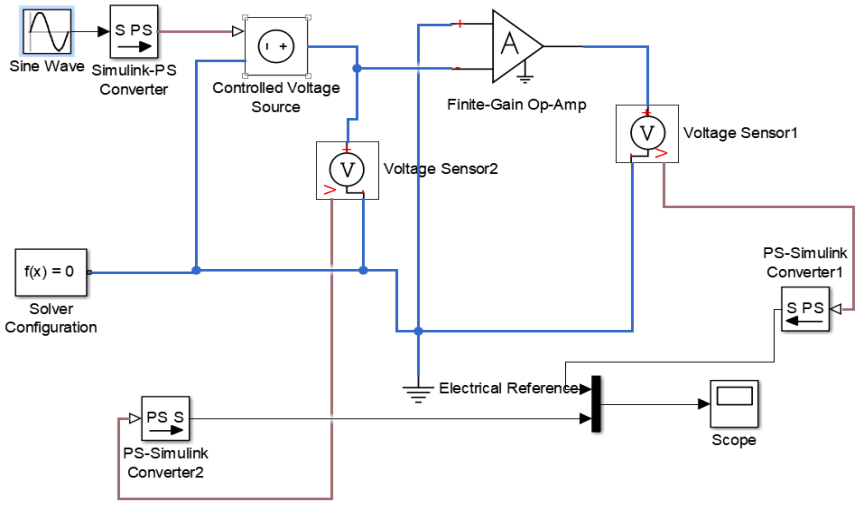
5.1.3 Comparador

O comparador implementado tem o objetivo de gerar uma onda quadrada de frequência igual ao tacogerador modulada para uma tensão máxima de 3,3V e mínima de 0 V, compatíveis com os níveis de tensão para entradas digitais do dispositivo FPGA do SOC-FPGA usado para processamento e cálculo de velocidade do motor.

O Comparador é feito com o AMP OP, de modo que o seu valor de saída satura para o valor de referência positivo de alimentação sempre que o sinal de entrada (vindo do tacogerador) na porta inversora é menor que o sinal de entrada da porta não inversora que é conectada ao terra, e da mesma forma no ciclo positivo do sinal de entrada do tacogerador o sinal de saída assume o valor de saturação da referência de alimentação negativa.

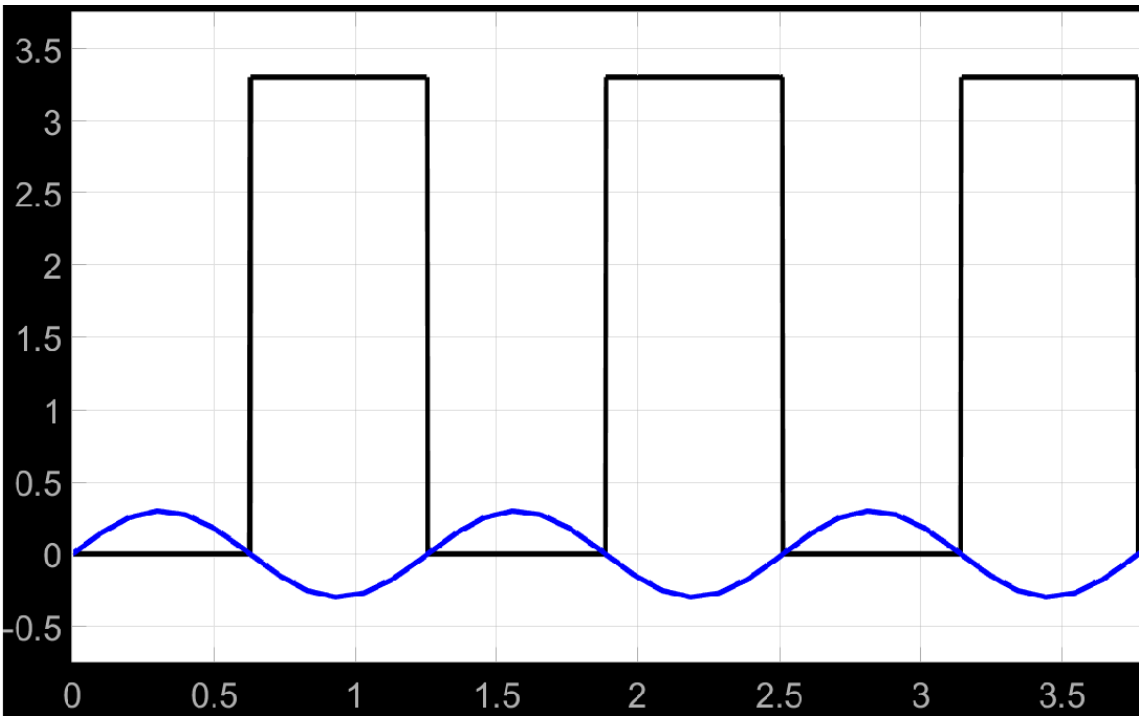
Na Figura 50 tem-se o esquemático do circuito no SIMULINK® e na Figura 51 as formas de ondas resultantes da conversão:

Figura 50 - Esquemático no Simulink do circuito que transforma uma onda senoidal em quadrada



Fonte: Próprio Autor

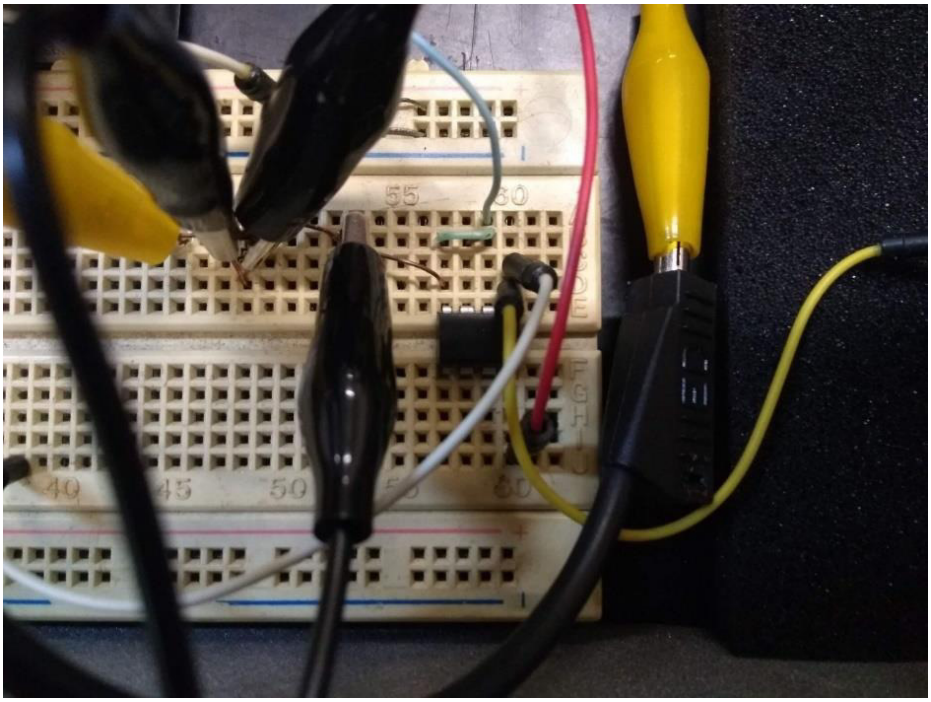
Figura 51 - Resultado da simulação para o esquemático da Figura 50



Fonte: Próprio Autor

Na Figura 52 tem-se o circuito real em protoboard e na Figura 53 é mostrado os sinais de entrada e saída do comparador para diferentes valores de velocidade. De modo que se observa que para diferentes níveis de tensão e frequência da entrada, sempre se tem uma saída de onda quadrada com nível de tensão oscilatório de 0 e 3,3V em frequência igual ao do tacogerador.

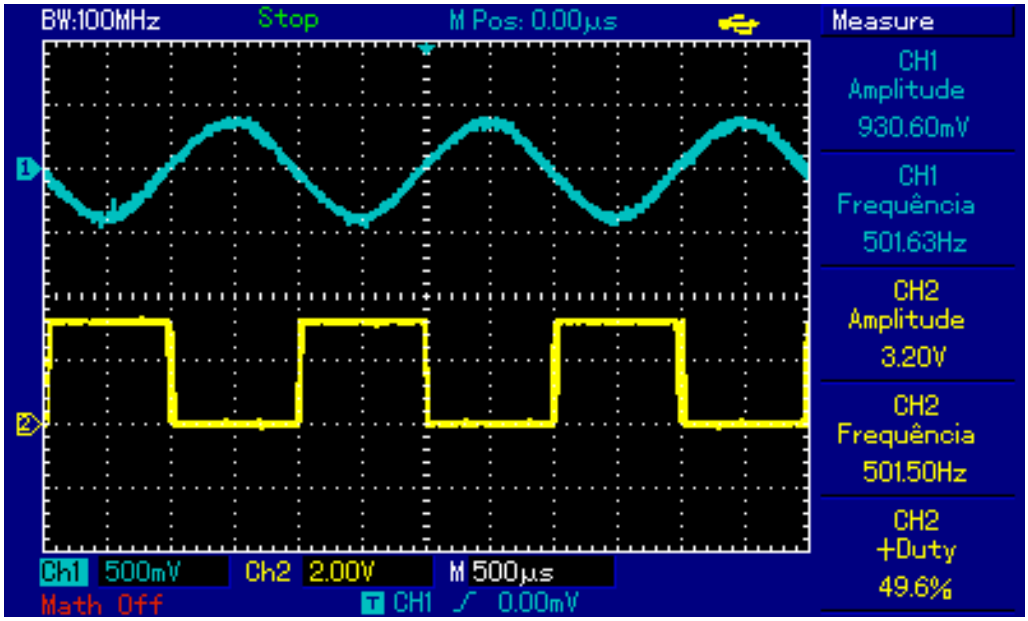
Figura 52 - foto do circuito que recebe o sinal do taco gerado e condiciona para a entrada do FPGA



Fonte: Próprio Autor

Na Figura 53 observa-se que em uma velocidade de 1000rpm do motor, para uma entrada de tensão senoidal de aproximadamente 930mV e frequência de 501Hz, tem-se uma saída com valor de 3,3, 0V e frequência de 501Hz, como de acordo com a simulação.

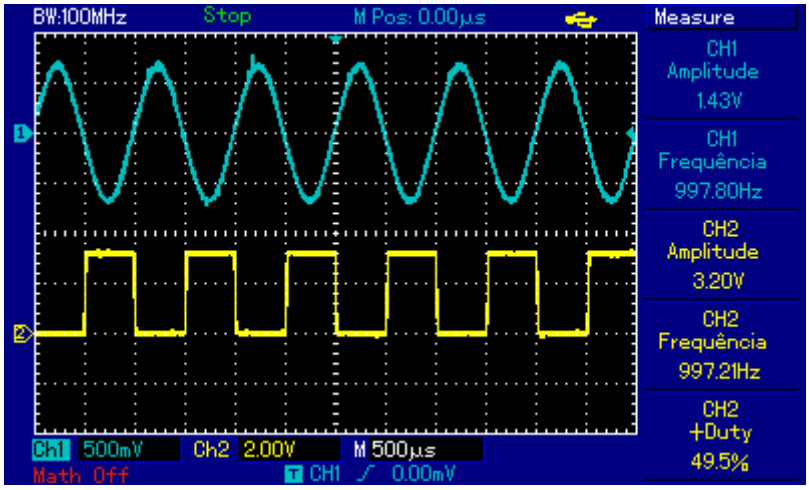
Figura 53 - Formas de onda do comparador para uma velocidade de 1000rpm



Fonte: Próprio Autor

Na Figura 54 observa-se que em uma velocidade de 2000rpm do motor, para uma entrada de tensão senoidal de aproximadamente 1,43V e frequência de 997,8Hz, tem-se uma saída com valor de 3,3, 0V e frequência de 997,8Hz , como de acordo com a simulação.

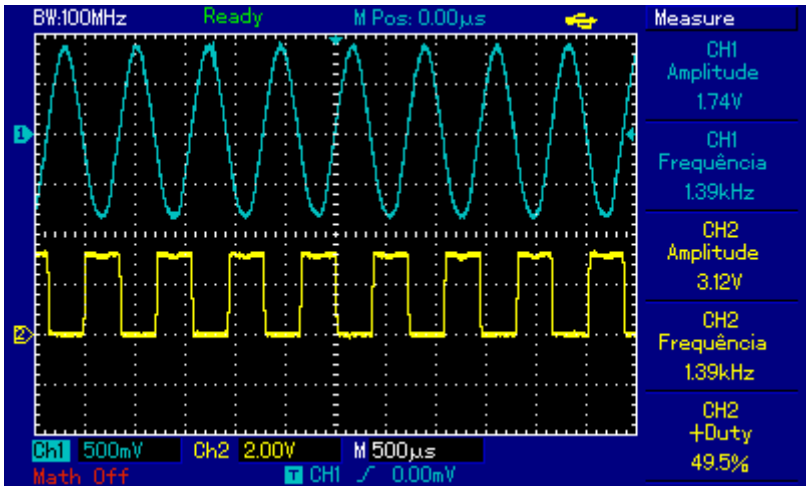
Figura 54 - Formas de onda do comparador para 2000rpm



Fonte: Próprio Autor

Na Figura 55 observa-se que em uma velocidade de 3000 do motor, para uma entrada de tensão senoidal de aproximadamente 1,74V e frequência de 1,39Khz, tem-se uma saída com valor de 3,3, 0V e frequência de 1,39Khz, como de acordo com a simulação.

Figura 55 - Formas de onda do comparador para 3000 rpm



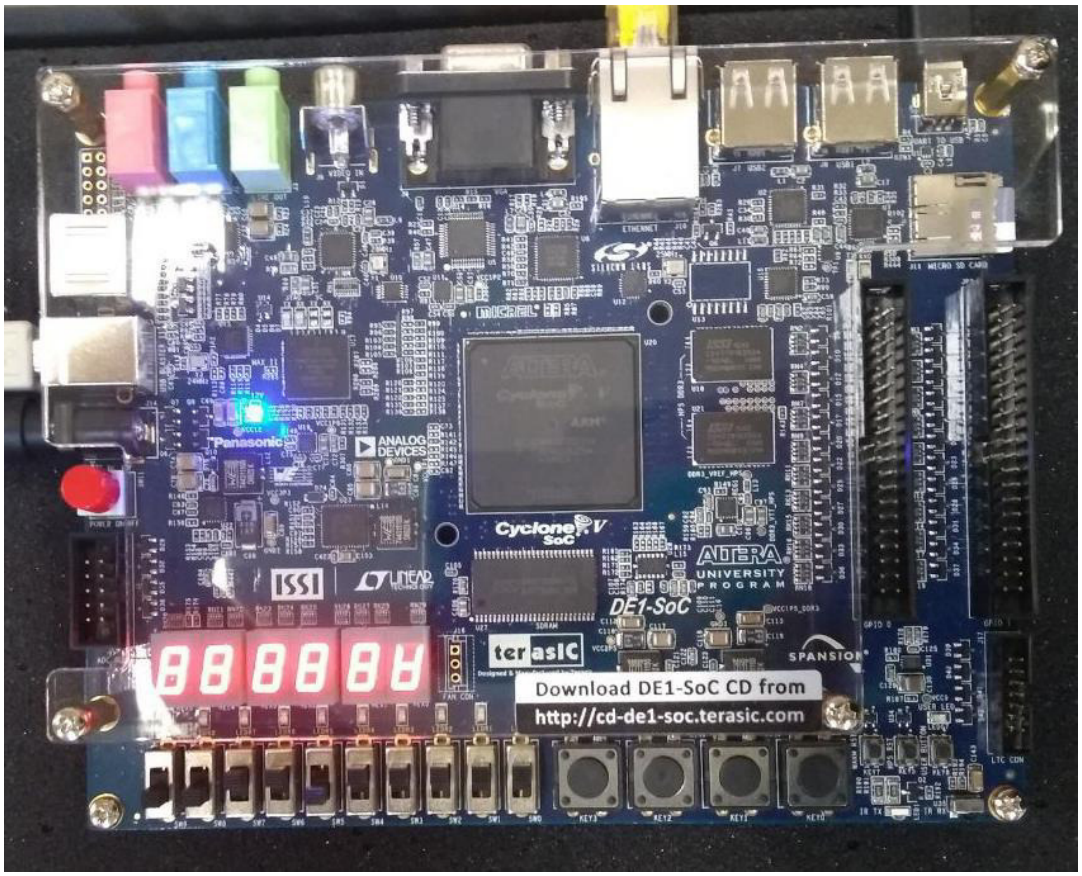
Fonte: Próprio Autor

5.1.4 SOC FPGA

Na placa SOC FPGA (Figura 56), possui dois níveis de programação. A parte do FPGA programada em Verilog e a parte do ARM programada em C com bibliotecas LINUX.

A parte do FPGA tem em resumo três funções básicas a primeira de sensoriamento de forma que faz a leitura de período do sinal de tacho, a segunda de geração de PWM com largura de pulso dada por software Arm e a terceira é a implementação de conexões e interfaceamentos entre o ARM® e o FPGA. Cada uma dessas implementações será especificada, a seguir

Figura 56 - Foto do módulo DE1-SoC FPGA usado no presente projeto

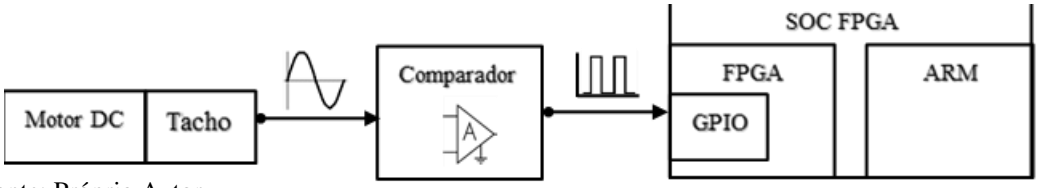


Fonte: Próprio Autor

5.1.4.1 Leitor de Sinal de Tacho

Como já discutido anteriormente para uma determinada velocidade o taco gera um sinal de frequência proporcional a do sinal do Motor que é condicionado pelo comparador e recebido pelo GPIO do FPGA do SOCFPGA. (Figura 57)

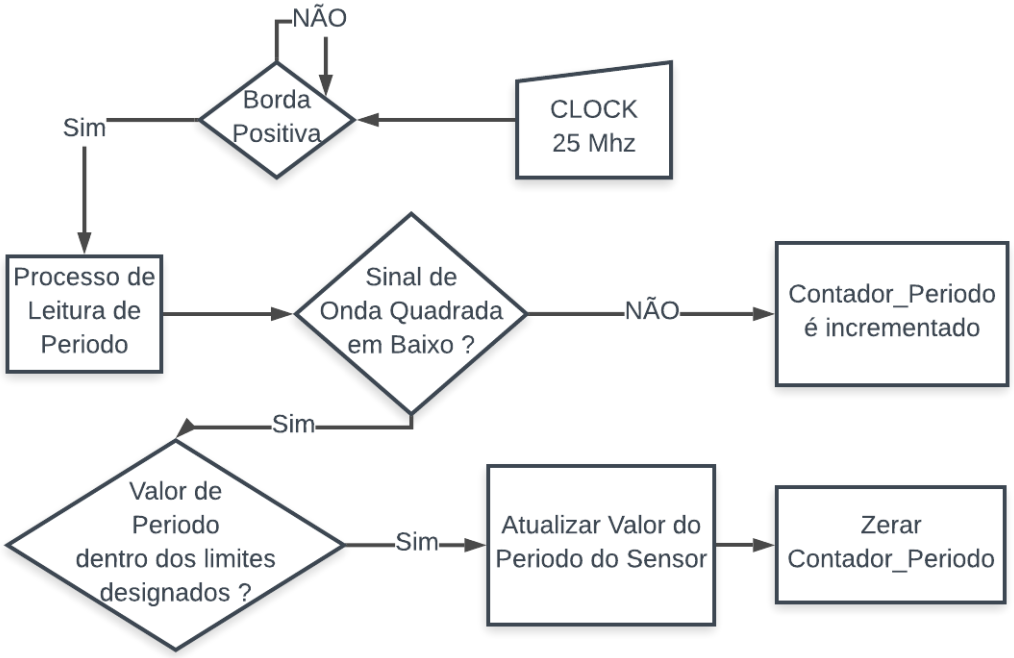
Figura 57 - Diagrama de blocos do fluxo de leitura da velocidade do motor.



Fonte: Próprio Autor

No FPGA se tem o código em Verilog para a leitura de Tempo em alta do sinal de entrada, de modo que a cada borda de subida de um clock de 25MHz é iniciado o processo de leitura em que se incrementado o valor do contador de período do sinal enquanto o sinal de entrada estiver em alto, e quando o sinal estiver em baixa o valor do período da borda contado durante o tempo em alta é atualizado em uma variável que fica disponível para leitura do ARM, de modo que depois que atualizado o valor da variável o contador de período é zerado, e como o contador de período não estar com valor dentre dos limites aceitáveis de período do sinal, o FPGA "não faz nada" até que ocorra novamente um nível alto na entrada para assim iniciar a contagem. Abaixo tem-se um fluxograma do algoritmo pensado para a leitura de Período do sinal de entrada (Figura 58)

Figura 58 - Fluxograma do algoritmo de leitura de período implementado no FPGA



Fonte: Próprio Autor

A Tabela 3 mostra alguns valores calculados e a Tabela 4 mostra os valores experimentais para diferentes sinais de PWM gerados. Testificando a precisão do contador de período do sinal de entrada quadrada. A frequência de contagem foi tida através das sucessivas divisões entre os processos do *Clock* de 50MHz.

Tabela 3 - Cálculo do Valor de contagem do contador de período para diferentes frequências

Sensor de Período de Tacho Gerador (Cálculo)	
Cont_contagem	**Freq_contagem**
2	4.166.667
F_Sinal(Hz)	**Valor_Contagem**
65	64.102,56
500	8.333,33
1000	4.166,67
1500	2.777,78
2000	2.083,33
3000	1.388,89

Fonte: Próprio Autor

Tabela 4 - Exemplo de duas amostras de valores de contagem para diferentes frequências de sinal de entrada

Sensor de Período de Tacho Gerador (Experimental)		
F_Sinal(Hz)	**Valor_Contagem**	
60Hz	69444	69444
65	64102	64102
500	8333	8333
800	5208	5208
1000	4166	4167
1200	3472	3472
1800	2315	2314
2000	2083	2084
3000	1389	1389
4	1041655	1041655
20	208332	208332
40	104165	104165

Fonte: Próprio Autor

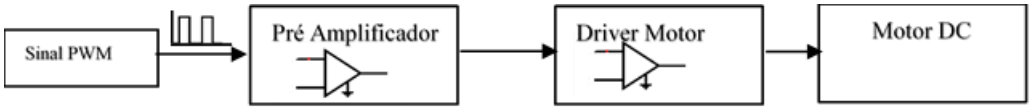
O valor de contagem de período fica todo tempo disponível para a leitura do ARM® que a cada interrupção programada “pega” o valor de período e calcula a saída “u” do controlador para a planta. A saída “u” é dada em largura de pulso de um PWM de 1KHz gerado em *Hardware*.

Para calcular o valor do *duty cycle* é deduzida a fórmula seguinte:

$$Duty(\%) = \frac{u}{50000} * 100\% \qquad (29)$$

Depois de calculado o valor de *duty cycle*, é gerado um sinal PWM que passa por drivers de amplificação até o motor DC, como é mostrado na figura 59.

Figura 59 - Diagrama em blocos do fluxo de sinal do PWM até a entrada do Motor

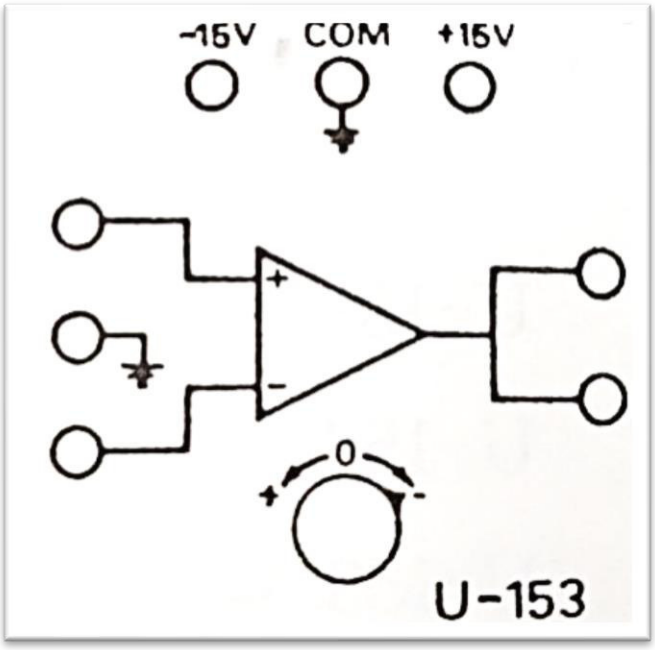


Fonte: Próprio Autor

5.1.5 Pré Amplificador

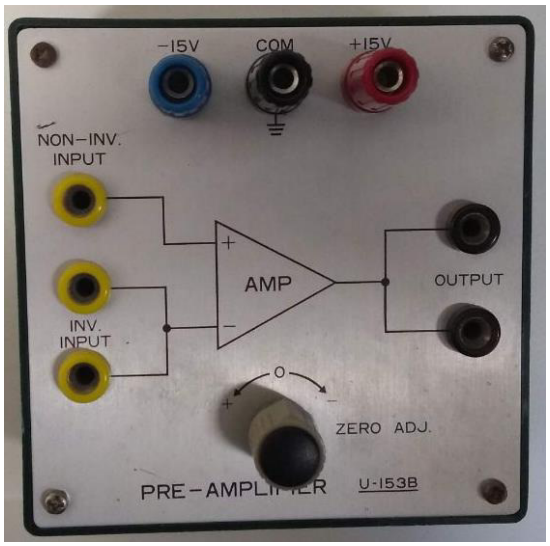
Depois de que o sinal sai do FPGA ele passa por dois estágios de amplificação, o primeiro estágio é o Pré-Amplificador que tem como função amplificar a tensão para os níveis de alimentação compatíveis com o Motor DC. Como pré-amplificador é utilizado o Módulo U-153, a ilustração (Figura 60) e a foto do módulo (Figura 61) é mostrada a seguir.

Figura 60 - Ilustração do módulo pré amplificador U-153



Fonte: (INIMIPA, 2004)

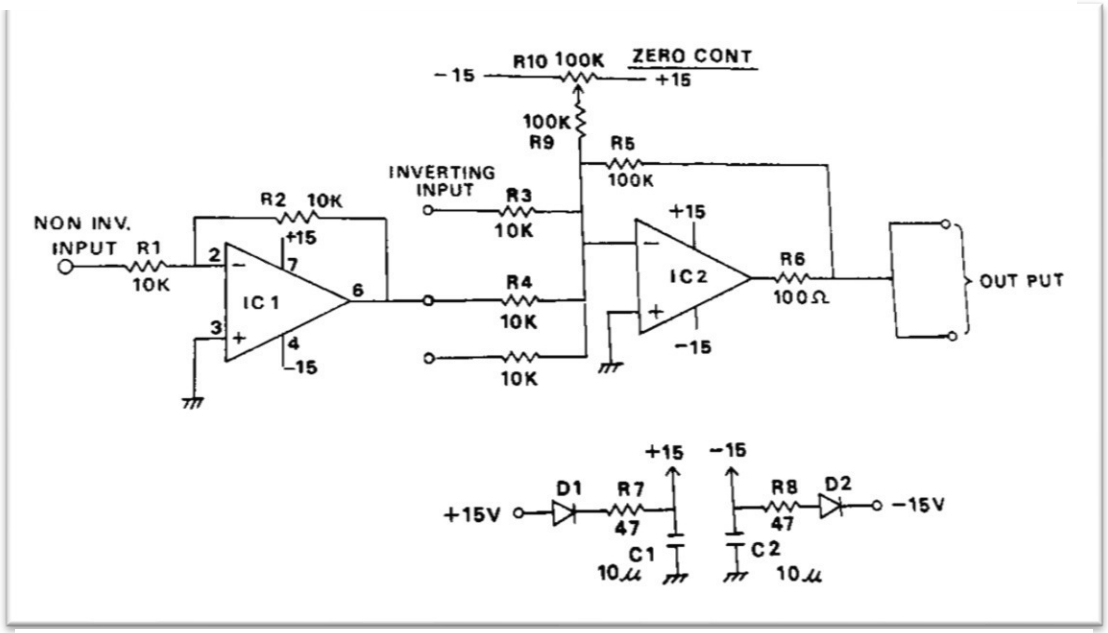
Figura 61 - Foto do módulo U153 usado no projeto



Fonte: Próprio Autor

O esquema da estrutura eletrônica do módulo é mostrado na Figura, observa-se que ele tem duas possibilidades para o sinal de entrada, tanto na entrada não inversora quanto na entrada inversora. Também se percebe que ele possui um potenciômetro para ajuste de offset da entrada, por fim observando o resistor de entrada e comparando com o de realimentação observa-se um ganho de 10.

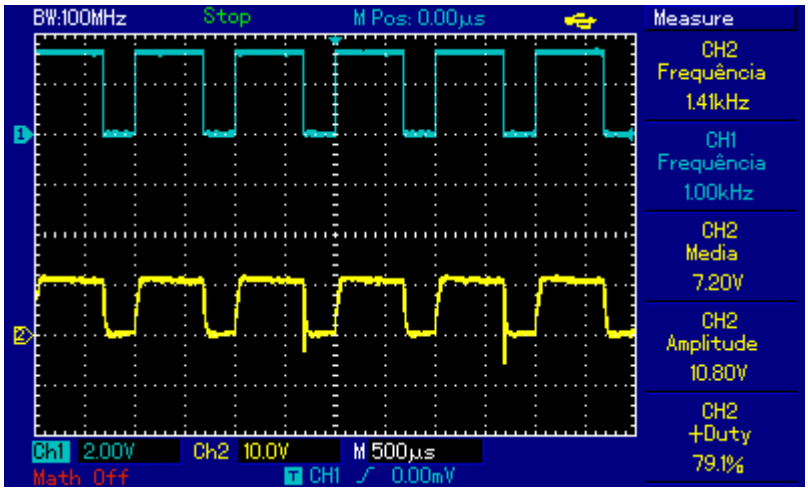
Figura 62 - Diagrama de ligação interna ao módulo pré amplificador



Fonte: (INIMIPA, 2004)

Na Figura 63, observa-se que a saída de PWM do FPGA é de um nível de tensão de 0 e 3,3V, essa tensão é então amplificada mantendo a mesma frequência com saturação dos amplificadores em torno de 11V.

Figura 63 - Sinais de amplificação da onda PWM, sinal de entrada (em cima) e sinal amplificado (em baixo)



Fonte: Próprio Autor

5.1.6 Driver Amplificador do Motor

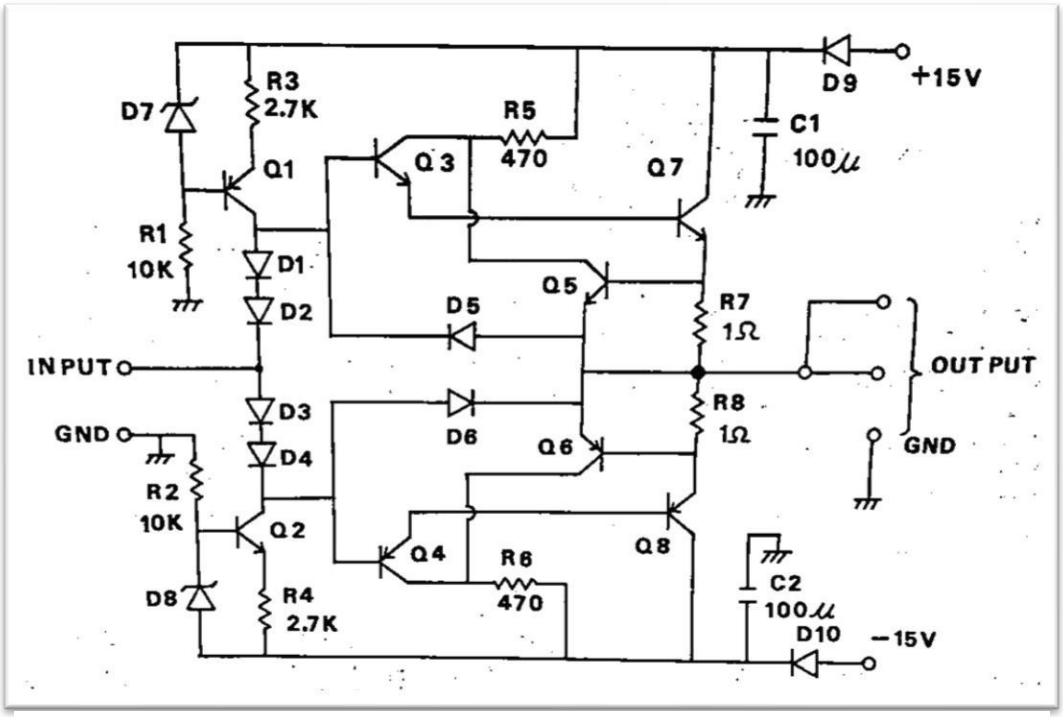
O segundo estágio de amplificação é o estágio de amplificação de corrente o qual tem uma alimentação independente da do resto do circuito, dedicada somente para o driver do motor. Para o driver do motor é usado o módulo Driver Amplificador Motor (Ganho: 0dB) U-154 (Figura 64).

Figura 64 - Foto do módulo Driver Amplificador Motor U154

Fonte: Próprio Autor

Na Figura 65 tem-se o esquema interno do circuito de amplificação de corrente alimentado com fonte externa simétrica. Como observa-se esse tipo de amplificação não tem o intuito de aumentar o nível de tensão, mas sua prioridade é aumentar a capacidade de aumento de corrente do circuito.

Figura 65 - Esquema eletrônico do módulo driver amplificador de corrente



Fonte: (INIMIPA, 2004)

5.2 Algoritmo e detalhamento de interfaceamento SOCFPGA

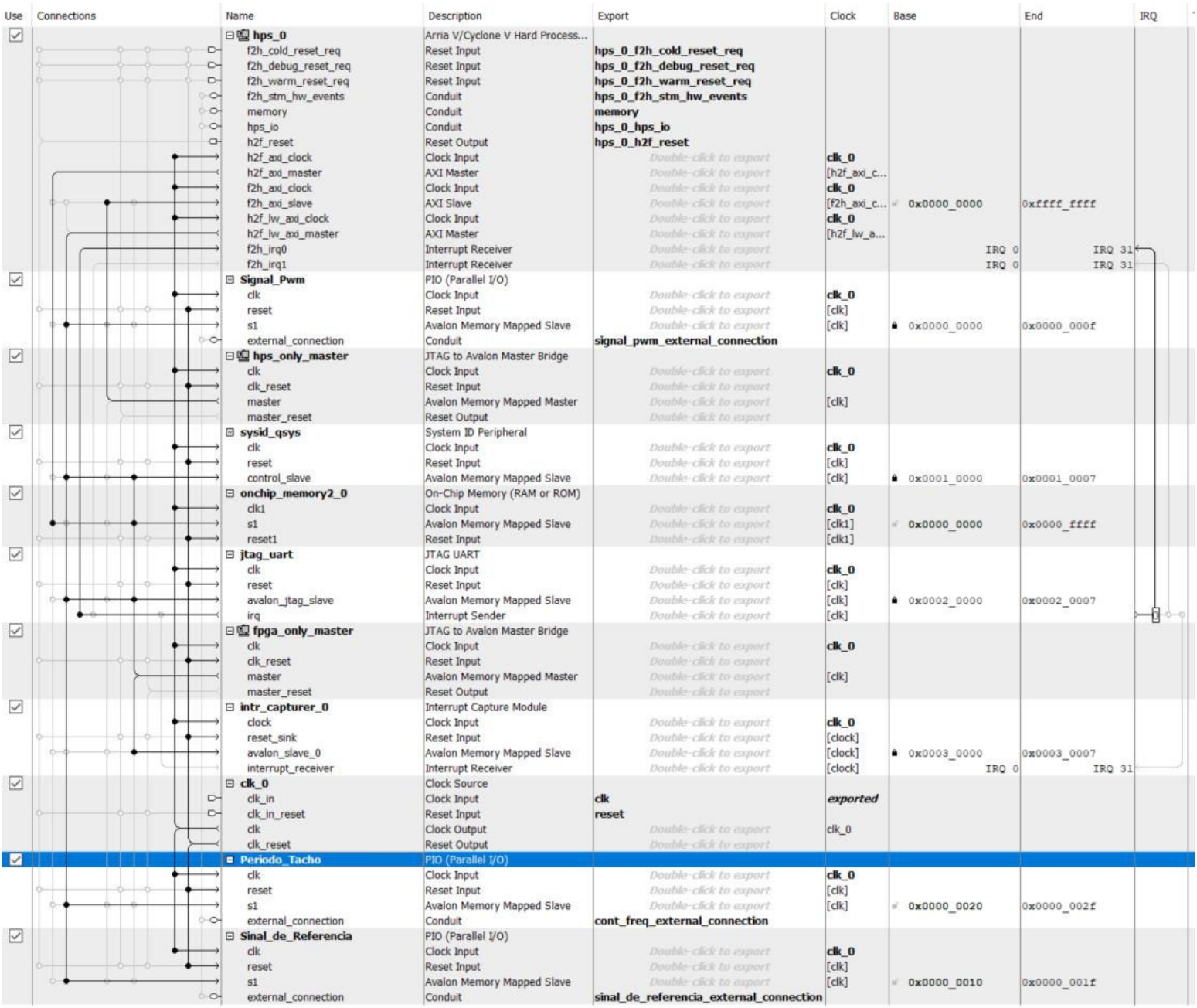
Todo o interfaceamento entre o FPGA e o ARM foi elaborado através da ferramenta *Qsys Tools* do software *Quartus II,* onde através dessa ferramenta se faz as ligações e configurações dos diversos controladores e barramentos configurando cada um de acordo com as especificações do processador e de acordo com as especificações do projeto, podendo escolher os tipos de interfaces, configurar os *clock*s, configurar as entradas, saídas e memórias utilizadas pelo HPS, além de configuras as interrupções, endereços entre outras funcionalidades.

Como se pode ver na Figura 66, a ferramenta Qsys tools dá suporte a importar cada tipo de periférico, controlador e ele é adicionado em linhas, em relação a primeira coluna temos a ativação do controlador ou periférico, a seguir temos na segunda coluna as conexões que são feitas entre os diversos *paths*, por conseguinte o nome dos dispositivos (em negrito) e portas (sequência de linhas abaixo do negrito), depois temos a descrição de cada, por conseguinte tem-se o *export* que é a parte da plataforma onde se tem as conexões externas, ou seja as conexões

e barramentos programáveis pelo usuário em HDL, a seguir se tem o endereço de base e ofsset que são a forma de endereçamento vista pelo HPS que posteriormente será acessada pelo software em C, por fim as interrupções que são enumerada e controladas pelo controlador de interrupção. (COSME, 2018)

Por fim em relação aos elementos de programação do *Qsys* temos o HPS Cyclone V, que foi instanciado como "*hps_0*", nele se programou as conexões de interface *AXI* e *LW* entre o Hps e o fpga (h2f e f2h), e também se tem a definição do *clock* instanciado como "*clk_0"*. As configurações dos barramentos de entrada e saída (*Signal_pwm, Periodo_Tacho, Sinal_Referencia*) são feitas através da escolha do tamanho de bits e configuração de direcionamento de bits, se vão ser entrada, saída ou bidirecionais (Entrada/Saída). Os outros elementos como *"on_chip_memory", "jtag_uart",* entre outros são relacionados ao modo de gravação do kit, outras organizações dos endereçamentos e organização das interrupções que não se vê muito necessário entrar em detalhes.

Figura 66 - Interfaceamento entre FPGA e ARM pela plataforma Qsys tools do Quartus



Fonte: Próprio Autor

Depois de o dispositivo SoC FPGA ter sido configurado pela plataforma Qsys, nela é gerada a instanciação de todos os módulos usados para mapeamento dentro do kit usado, ou seja, depois de instanciado é direcionado e localizado cada sinal e conexão dentro do kit do SoC FPGA escolhido, como se fosse fazer um grande *pin_planner*, para os que já estão acostumados com progração de FPGAs.

No anexo B, observa-se parte do código de instanciação dos elementos de Hardware no FPGA, onde dentro dos "()" em *verilog* se tem o elemento de *hardware* que foi programado, elemento esse que pode ser um sinal, variável ou até conexão física de fábrica.

Todos esses endereçamentos são feitos com o auxílio do manual do kit do fabricante, parte dele é mostrado na Figura 67.

Figura 67 - Parte do Manual onde se encontra o endereçamento e localizações dos diversos pinos dos dispositivos do SoC FPGA

Table 3-27 Pin Assignment of DDR3 Memory

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
HPS_DDR3_A[0]	PIN_F26	HPS DDR3 Address[0]	SSTL-15 Class I
HPS_DDR3_A[1]	PIN_G30	HPS DDR3 Address[1]	SSTL-15 Class I
HPS_DDR3_A[2]	PIN_F28	HPS DDR3 Address[2]	SSTL-15 Class I
HPS_DDR3_A[3]	PIN_F30	HPS DDR3 Address[3]	SSTL-15 Class I
HPS_DDR3_A[4]	PIN_J25	HPS DDR3 Address[4]	SSTL-15 Class I
HPS_DDR3_A[5]	PIN_J27	HPS DDR3 Address[5]	SSTL-15 Class I

Fonte: (TERASIC TECHONOLOGIES, 2015)

5.3 Controlador PI

5.3.1 Equação do Controlador e algoritmo de implementação

Para esse projeto, como o foco era o desbravamento e a integração entre o FPGA e o ARM® optou-se por medida de simplificação não fazer formalmente a identificação da planta e fazer o projeto no domínio do tempo observando a resposta da planta para diferentes valores de Kp e Ki e aplicando a equação PI discretizada.

Como já demonstrado na teoria de controle PI, a equação de controle utilizada foi a (15), porém sem o termo derivativo:

$$u[k] = u[k-1] + K_p * (e[k] - e[k-1]) + K_i T * e[k] \qquad (29)$$

Ou essa equação também pode ser escrita isolando os erros, que ficaria da forma:

$$u[k] = u[k-1] + \left(K_i T * K_p\right) * e[k] - K_p * e[k-1] \qquad (30)$$

Porém como o sinal de y[k] é um período, se for desejado aumentar o y[k] se deve diminuir o valor de u[k], ou seja, diminuir a tensão no motor diminuído a sua velocidade e consequentemente aumentando o período. Portanto a equação deve ser reescrita da forma:

$$u[k] = u[k-1] - K_p * (e[k] - e[k-1]) - K_i T * e[k] \qquad (31)$$

Para explicar melhor considere o exemplo:

Considere uma referência de 200ms, se o sinal de sensor receber um valor de 400ms indica que a velocidade do motor estar abaixo do que se deseja na referência, ou seja, é

necessário que o sinal no controlador aumente, por tanto conclui-se que os parâmetros de controle deve atuar aumentando o valor de u[k], para o controle PI observa-se:

Analisando a equação de controle, se o e[k] > 0, ou seja, o valor da referência for maior que ao da saída de período

$$e[k] > 0 \rightarrow e[k] = r - y[k] > 0 \rightarrow r > y[k]$$

Então o valor de y[k] (período do motor) precisa diminuir, para se ter isso o valor de u[k] (largura de pulso do pwm) deve aumentar, portanto u[k] deve ser maior do que u[k-1], mostrando a seguinte relação:

$$Para\ e(k) > 0 \rightarrow u[k] > u[k-1].$$

Como a equação de Integração tem-se que

$$u[k] = u[k-1] + TKi * e[k]$$

Observa-se que o controlador não deve atuar incrementando o sinal anterior caso o erro seja maior do que zero, mas sim o inverso, portanto por conta de que para diminuir a saída precisa-se aumentar a entrada, ou seja:

tem-se que a equação fica reescrita como:

$$u[k] = u[k-1] - Kp(e[k] - e[k-1]) - TK_i e[k]$$

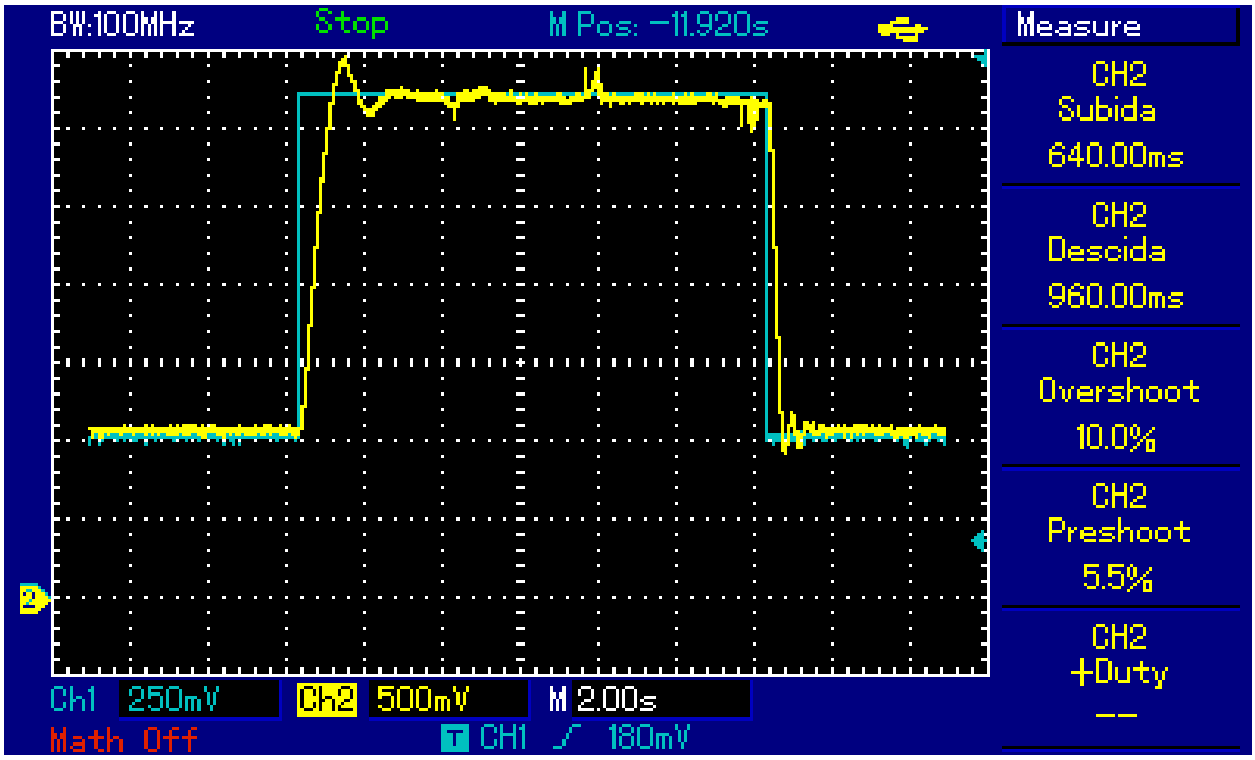
No Anexo A, é demonstrada a parte do código em C onde é feito o controle PI, observa-se que todo o controle é feito dentro de uma rotina de interrupção programada com frequência fixa, e que além das atribuições simples de controle foi-se necessário atuar com alguns saturadores para não dar "*overflow"* nas variáveis e nem permitir com que fosse entregue ao motor um sinal de velocidade maior do que a suportado.

5.3.2 Resultado dos Sinais Controle

A partir da variação dos valores de Kp e Ki, de modo que observando as respostas do regime transitório e *overshoot* do sinal de velocidade, foi-se obtido diferentes respostas para diferentes valores desses parâmetros. A seguir tem-se algumas formas de ondas para alguns valores de Kp e Ki escolhidos. O objetivo do controle era obter uma velocidade que seguisse um sinal de referência escrito no próprio código, a partir disso trabalhou-se para ter o sinal mais próximo do de referência.

Na Figuras 68 observa-se que a resposta ao degrau estava seguindo bem a referência, porém apresentava um *overshoot* considerável.

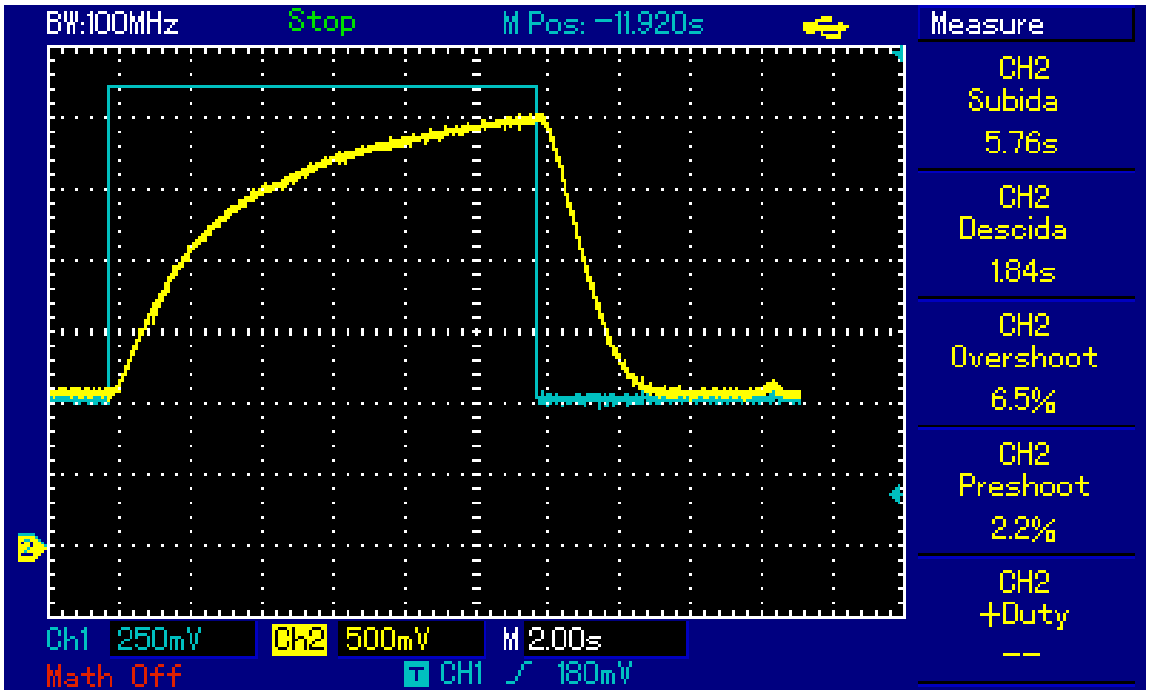
Figura 68 - Resposta ao degrau para os valores de "Kp" e Ki" iguais a 1



Fonte: Próprio Autor

Na Figuras 69 observa-se que a resposta ao degrau estava tão lenta que para nem chegava ao sinal de referência dentro do intervalo de tempo posto pelo projetista.

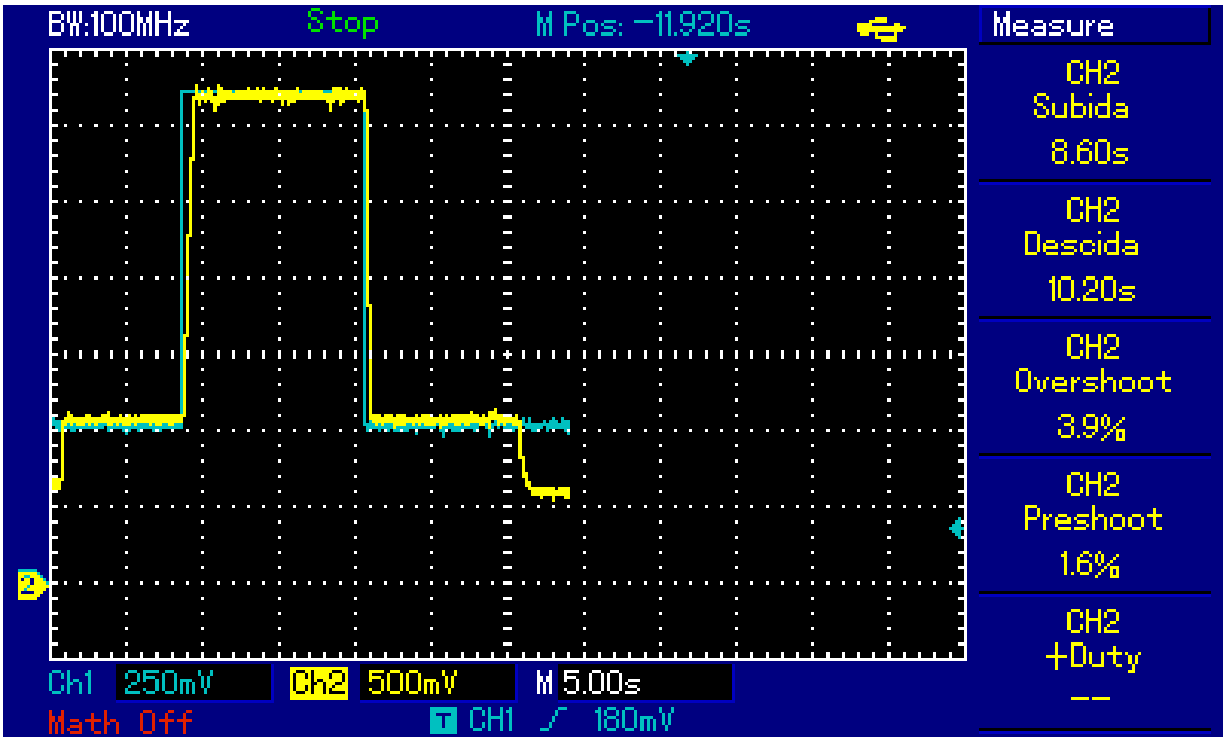
Figura 69 - Resposta ao degrau para os valores de "Kp= 0.05" e "Ki = 0.02"



Fonte: Próprio Autor

Na Figuras 70 observa-se que a resposta ao degrau seguiu rapidamente e sem *oveshoots* a referência dada, tanto para uma referência de aumento quanto de diminuição de velocidade. Portanto se viu como os melhores parâmetros do controlador.

Figura 70 - Resposta ao degrau para os valores de "Kp= 11" e "Ki = 10"



Fonte: Próprio Autor

6. CONCLUSÃO

Através desse trabalho conclui-se que o SOC FPGA, possui grande liberdade de programação e apresenta um projeto de hardware de forma muito completa, isso foi muitas vezes observado na precisão e exatidão dos sinais PWMs gerados com largura *duty cycle* correspondendo aos valores projetados sem oscilações além também da precisão tida na leitura de período do sinal em que se foi testado através de geradores de frequência fixa, e a leitura através dos contadores na parte FPGA foram bastante satisfatórios.

Um dos desafios foi o estudo do funcionamento de um sistema operacional e suas bibliotecas, além do mapeamento dos periféricos no FPGA dentro do Linux, porém depois de estudado e implementado essa etapa a possibilidade de carregamento de um sistema operacional trouxe facilidades em relação a alteração de valores, monitoramento de variáveis, exportação de dados, comunicação com os outros periféricos e muitas outras funcionalidades que vêm da programação e desenvolvimento de aplicações em uma linguagem de mais alto nível.

A implementação de uma lei de controle depois de programadas todas as bases de hardware e software foi uma das partes mais simples do projeto, de modo que houve apenas algumas dificuldades em relação a manipulação de variáveis pois em algumas situações, por conta dos valores de ação integral e proporcional muito grande, havia *overflow* das variáveis, porém depois de descoberto o problema a implementação da solução na linguagem C foi bem mais simples do que se estivesse em HDL, de modo que as bibliotecas em C para o Linux oferecem muitas funções e variáveis de diferentes tipos e tamanhos, bastando ao programador conhece-las e respeitar seus limites.

Portanto, conclui-se que a possibilidade de carregar um sistema operacional em um microcontrolador e ter essa integração com o FPGA tornou o uso da arquitetura FPGA muito mais viáveis pois algoritmos que são muitas vezes extensos e complexos de serem implementados em HDL são facilmente implementados com o uso das bibliotecas em C e nas aplicações para o Linux. O desenvolvimento de controladores mais precisos e a grande possibilidade de programação concorrente em hardware oferecida pelo FPGA é muito mais completa quando em conjunta com uma programação sequencial e estruturada de microcontroladores, de modo que a combinação desses dois grandes poderes de processamentos distintos torna o SoC FPGA um dispositivo de altíssimo desempenho e versatilidade na solução de problemas de engenharia.

Este trabalho pode ser suporte para diversas aplicações à posterior, como o desenvolvimento de hardware e firmares para controladores de pontes inversoras trifásicas que

é necessário a geração, dependendo da aplicação, em torno de 12 PWMs, também pode ser realizado um sistema de aquisição de dados de uma rede de sensores, em que no FPGA é feito o interfaceamento e hardware para receber valores de diversos tipos de sensores em uma indústria, sistema de irrigação, veículo, sistema de geração de energia, sistema de monitoramento e vários tipos de sistemas, em que todos esses dados recebidos pelo FPGA podem ser carregados em nuvem através de bibliotecas e ferramentas usadas no processador Linux, entre muitas outros tipos de aplicações.

7. REFERÊNCIAS

ALTERA. **Cyclone V Device Handbook**. San Jose: [s.n.], v. 3: Hard Processor System Technical Reference Manual, 2012.

ALTERA CORPORATION. **Architecture Brief, What is in a SoC FPGA**. San Jose, p. 4. 2014.

ANDRADE, H. P. **Sensor de velocidade**. Natal: [s.n.], 2008.

ARM. Architecture, Processors, and Devices. **Development Article**, 19 Maio 2009. 12.

BRUSCHI, S. M. **Arquitetura de Computadores SSC0114**. São Paulo: [s.n.], 2009.

CHAPMAN, S. J. **Fundamentos de máquinas elétricas**. 5. ed. Porto Alegre: ABDR, 2013.

COSME, E. P. **IMPLEMENTACÃO DE UMA ESTRATÉGIA DE MODULAÇÃO PWM SENOIDAL UTILIZANDO HARD PROCESSOR SYSTEM DE UM DISPOSITIVO SOC FPGA**. única. ed. Fortaleza: UFC, v. 1, 2018.

CURRY, D. A. **UNIX Systems Programming for SVR4**. 3. ed. Sebastopol: A nutshell handbook, v. 1, 2014.

DORF, R. C.; BISHOP, R. H. **Sistemas de Controle Modernos**. 8. ed. Rio de Janeiro: LTC, v. único, 2001.

INIMIPA. **Sistema de Treinamento em Servo-Mecanismo ED-4400B**. São Paulo: [s.n.], 2004.

INTEL. Intel® FPGAs and Programmable Devices. **Site da Intel**, 2019. Disponivel em: <https://www.intel.com/content/www/us/en/products/programmable/fpga.html>. Acesso em: 03 junho 2019.

INTEL CORPORATION. **Intel User-Customizable SoC FPGAs**. [S.l.], p. 19. 2017.

LAGES, W. F. **Modelagem de Sistemas Discretos**. Rio Grande do Sul: [s.n.], 2011.

MORENO, R. H. **Estudo para discretização de controladores PIDs Industriais**. Curitiba: UTFPR, 2011.

NISE, N. S. **Engenharia de Sistemas de controle**. 6. ed. Rio de Janeiro: LTC, v. único, 2012.

OGATA, K. **Engenhria de controle Moderno**. Rio de Janeiro: PGB, 1985.

PALNITKAR, S. **A guide to Digital Design and Synthesis**. 1. ed. [S.l.]: Prentice Hall PTR, v. único, 1996.

PICCIONI, C. A.; TATIBANA, C. Y.; DE OLIVEIRA, R. S. **Trabalhando com o Tempo Real em Aplicações Sobre o Linux**. UFSC. Florianópolis, p. 66. 2001. (CTC/UFSC).
PRADO, L. F. P. D. P. Uma Avaliação do processo de protabilidade do sistema operacional android para uma plataforma embarcada. **Monografia**, São Carlos, v. 1, n. 1, p. 59, Dezembro 2011. ISSN EESC/USP.

RAMALHO, L. A. Uso de linguagem de descrição de hardware e dispositivos de alto desempenho na educação tecnológica. **Jornada de Pesquisa e Exentenção**, Cuiabá, p. 10, 2013.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. **Sistemas Operacionais com Java**. 1. ed. [S.l.]: Campus, 2016.

TEIXEIRA, P. R. F.; PETROBRAS. **Instrumentista de sistemas - Fundamentos de controle**. Rio de Janeiro: Promin, 2006.

TERASIC TECHONOLOGIES. In: PROGRAM, A. U. **DE1-SoC - User Manual**. [S.l.]: [s.n.], 2015. p. 116.

TORO, V. D. **Fundamentos de máquinas elétricas**. [S.l.]: LTC.

WEBER, A. F. Arquitetura FPGAs e CPLDs da ALTERA, 2016.

APÊNDICE A: CÓDIGO EM C PARA O LINUX/ARM

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#include "hwlib.h"
#include "socal/socal.h"
#include "socal/hps.h"
#include "socal/alt_gpio.h"
#include "hps_0.h"
#include <math.h>

#include <signal.h>
#include <inttypes.h>
#include "alt_clock_manager.h"
#include "alt_generalpurpose_io.h"
#include "alt_timers.h"
#include "socal/alt_rstmgr.h"

#include"mytimer.h"

#define HW_REGS_BASE ( ALT_STM_OFST )
#define HW_REGS_SPAN ( 0x04000000 )
#define HW_REGS_MASK ( HW_REGS_SPAN - 1 )

uint32_t pwm_mask;
uint8_t ref_mask = 60;
int flag = 0;
int sentido = 0;
int cont_tacho;
int cont_pulso = 0;
void *h2p_cont_addr;
void *h2p_pwm_addr;
void *h2p_ref_addr;
int comutador = 0;
int loop_count;

int32_t e[2] = {0,0};
int32_t r = 8000; // quanto mais baixo mais r□◌do valor central = 6000,
valor baixo = 7000
uint32_t y[2] = {8000,8000};
uint32_t u[3] = {12000,12000,12000};

float Kp = 11;//0,16//0,01
float Ki = 10;//1

int32_t P = 0;
int32_t I = 0;
uint32_t n = 0;

// Rotina de interrup磯 para o controlador PI
void handler(int sig) {
    flag = 1;
// Equa磯 PID
// u[n]=u[n-1]+K_p*(e[n]-e[n-1])+K_i*T(e[n])+K_d/T (e[n]-2e[n-1]+e[n-2])
// Aquisi磯 e C□ulo das novas amostras
    y[1] =*(uint32_t *)h2p_cont_addr; // Sa(Periodo atual do motor)
    e[1] = r - y[1];                  // Calculando erro de agora
    P =  Kp*(e[1]-e[0]);             //Calculo do Termo Proporcional
```

```
    I =  Ki* e[1];                  // Calculo do Termo Integral

/* "Saturador" para diminuir os termos PI, para n棒ultrapassar u[0]
se n棒o valor de u ficaria negativo, pois u[1] = u[0] - P - I;
*/
    if ((I + P) < u[0]){
    u[1] = u[0] - P - I;
    } else {
        u[1] = u[0] - ((I+P)*0.8); // Caso Kp e Ki muito grande
    }
    // Atribui磯 das Amostras antigas
    y[0] = y[1];
    e[0] = r - y[0];

// Saturadores para n棒ultrapassar os limites de velocidade do motor
    if (u[1]<6000){
        pwm_mask = 6000;
        u[0] = 6000;
    }
    if (u[1]>45000){
        pwm_mask = 45000;
        u[0] = 45000;
    }
// Caso o valor de esteja dentro dos limites atribuir ao controlador
    if (u[1] > 6000 && u[1] < 45000){ // saturador
        pwm_mask = u[1];
        u[0] = u[1];
    }
// Prints dos valores para monitoramentos das vari‘is pelo usu□o
    if(loop_count % 25 == 0){
    printf ("%u %u %d %d %d \n",u[1],y[1],e[1],P, I);
    }
// Mudan硯dos degraus de refer□ia
    if(loop_count  == 1){// por 1
        r = 8333; // por 8333
        ref_mask = 60;
    }
    if(loop_count  == 800){// por 800
        r = 2777; // por 2777
        ref_mask = 255;
    }
    if(loop_count  == 2000){// por 2000
        r = 8333; // por 8333
        ref_mask = 60;
    }
    loop_count++;
}


int main() {
    printf(
    void *virtual_base;
    int fd;


    if ((fd = open("/dev/mem", ( O_RDWR | O_SYNC))) == -1) {
        printf("ERROR: could not open \"/dev/mem\"...\n");
        return (1);
    } else
```

```
        printf("memoria open w/sucess!\n");
    virtual_base = mmap( NULL, HW_REGS_SPAN, ( PROT_READ | PROT_WRITE),
            MAP_SHARED, fd, HW_REGS_BASE);
    if (virtual_base == MAP_FAILED) {
        printf("ERROR: mmap() failed...\n");
        close(fd);
        return (1);
    } else
        printf("mmap() open w/sucess!\n");

    h2p_pwm_addr = virtual_base
            + ((unsigned long) ( ALT_LWFPGASLVS_OFST + SIGNAL_PWM_BASE)
                    & (unsigned long) ( HW_REGS_MASK));
    h2p_cont_addr = virtual_base
                + ((unsigned long) (ALT_LWFPGASLVS_OFST +
PERIODO_TACHO_BASE)
                        & (unsigned long) ( HW_REGS_MASK));
    h2p_ref_addr = virtual_base
                    + ((unsigned long) ( ALT_LWFPGASLVS_OFST +
SINAL_DE_REFERENCIA_BASE)
                            & (unsigned long) ( HW_REGS_MASK));

//**********************Configura磯 da Interrup磯**********************
    pwm_mask= 15000;// 10 000 foi ate 2300 rpm // era 15000
    *(uint32_t *) h2p_pwm_addr = pwm_mask;
    *(uint32_t *) h2p_ref_addr = ref_mask;
    sleep(3);
    printf("u[1] y[1] e[1] \n");
    //loop_count = 2000;// POR 600

    signal(SIGALRM, handler);
    ualarm(1, 10000); // interrup磯 a cada 100us -> 10k

    while (loop_count < 3000) {//por 1800
        *(uint32_t *) h2p_pwm_addr = pwm_mask;
        *(uint32_t *) h2p_ref_addr = ref_mask;
    }
    *(uint32_t *) h2p_pwm_addr = 15000; // 10 000
    if (munmap(virtual_base, HW_REGS_SPAN) != 0) {
        printf("ERROR: munmap() failed...\n");
        close(fd);
        return (1);
    }
    close(fd);
    return (0);
}
```

APENDICE B: CÓDIGO EM VERILOG DO FPGA

```
`define ENABLE_HPS

module ghrd_top(

      ///////// ADC /////////
      inout              ADC_CS_N,
      output             ADC_DIN,
      input              ADC_DOUT,
      output             ADC_SCLK,

      ///////// AUD /////////
      input              AUD_ADCDAT,
      inout              AUD_ADCLRCK,
      inout              AUD_BCLK,
      output             AUD_DACDAT,
      inout              AUD_DACLRCK,
      output             AUD_XCK,

      ///////// CLOCK2 /////////
      input              CLOCK2_50,

      ///////// CLOCK3 /////////
      input              CLOCK3_50,

      ///////// CLOCK4 /////////
      input              CLOCK4_50,

      ///////// CLOCK /////////
      input              CLOCK_50,

      ///////// DRAM /////////
      output      [12:0] DRAM_ADDR,
      output      [1:0]  DRAM_BA,
      output             DRAM_CAS_N,
      output             DRAM_CKE,
      output             DRAM_CLK,
      output             DRAM_CS_N,
      inout       [15:0] DRAM_DQ,
      output             DRAM_LDQM,
      output             DRAM_RAS_N,
      output             DRAM_UDQM,
      output             DRAM_WE_N,

      ///////// FAN /////////
      output             FAN_CTRL,

      ///////// FPGA /////////
      output             FPGA_I2C_SCLK,
      inout              FPGA_I2C_SDAT,

      ///////// GPIO /////////
      inout     [35:0]         GPIO_0,
      inout     [35:0]         GPIO_1,


      ///////// HEX0 /////////
      output      [6:0]  HEX0,
```

```
        ///////// HEX1 /////////
        output      [6:0]  HEX1,

        ///////// HEX2 /////////
        output      [6:0]  HEX2,

        ///////// HEX3 /////////
        output      [6:0]  HEX3,

        ///////// HEX4 /////////
        output      [6:0]  HEX4,

        ///////// HEX5 /////////
        output      [6:0]  HEX5,

`ifdef ENABLE_HPS
        ///////// HPS /////////
        inout              HPS_CONV_USB_N,
        output     [14:0] HPS_DDR3_ADDR,
        output     [2:0]  HPS_DDR3_BA,
        output             HPS_DDR3_CAS_N,
        output             HPS_DDR3_CKE,
        output             HPS_DDR3_CK_N,
        output             HPS_DDR3_CK_P,
        output             HPS_DDR3_CS_N,
        output     [3:0]  HPS_DDR3_DM,
        inout      [31:0] HPS_DDR3_DQ,
        inout      [3:0]  HPS_DDR3_DQS_N,
        inout      [3:0]  HPS_DDR3_DQS_P,
        output             HPS_DDR3_ODT,
        output             HPS_DDR3_RAS_N,
        output             HPS_DDR3_RESET_N,
        input              HPS_DDR3_RZQ,
        output             HPS_DDR3_WE_N,
        output             HPS_ENET_GTX_CLK,
        inout              HPS_ENET_INT_N,
        output             HPS_ENET_MDC,
        inout              HPS_ENET_MDIO,
        input              HPS_ENET_RX_CLK,
        input      [3:0]  HPS_ENET_RX_DATA,
        input              HPS_ENET_RX_DV,
        output     [3:0]  HPS_ENET_TX_DATA,
        output             HPS_ENET_TX_EN,
        inout      [3:0]  HPS_FLASH_DATA,
        output             HPS_FLASH_DCLK,
        output             HPS_FLASH_NCSO,
        inout              HPS_GSENSOR_INT,
        inout              HPS_I2C1_SCLK,
        inout              HPS_I2C1_SDAT,
        inout              HPS_I2C2_SCLK,
        inout              HPS_I2C2_SDAT,
        inout              HPS_I2C_CONTROL,
        inout              HPS_KEY,
        inout              HPS_LED,
        inout              HPS_LTC_GPIO,
        output             HPS_SD_CLK,
        inout              HPS_SD_CMD,
        inout      [3:0]  HPS_SD_DATA,
        output             HPS_SPIM_CLK,
        input              HPS_SPIM_MISO,
```

```
      output              HPS_SPIM_MOSI,
      inout               HPS_SPIM_SS,
      input               HPS_UART_RX,
      output              HPS_UART_TX,
      input               HPS_USB_CLKOUT,
      inout       [7:0]   HPS_USB_DATA,
      input               HPS_USB_DIR,
      input               HPS_USB_NXT,
      output              HPS_USB_STP,
`endif /*ENABLE_HPS*/

      ///////// IRDA /////////
      input               IRDA_RXD,
      output              IRDA_TXD,

      ///////// KEY /////////
      input       [3:0]   KEY,

      ///////// LEDR /////////
      output      [9:0]   LEDR,

      ///////// PS2 /////////
      inout               PS2_CLK,
      inout               PS2_CLK2,
      inout               PS2_DAT,
      inout               PS2_DAT2,

      ///////// SW /////////
      input       [9:0]   SW,

      ///////// TD /////////
      input               TD_CLK27,
      input      [7:0]    TD_DATA,
      input               TD_HS,
      output              TD_RESET_N,
      input               TD_VS,


      ///////// VGA /////////
      output      [7:0]   VGA_B,
      output              VGA_BLANK_N,
      output              VGA_CLK,
      output      [7:0]   VGA_G,
      output              VGA_HS,
      output      [7:0]   VGA_R,
      output              VGA_SYNC_N,
      output              VGA_VS


);


// internal wires and registers declaration
  wire [3:0]  fpga_debounced_buttons;
  wire [9:0]  fpga_led_internal;
  wire        hps_fpga_reset_n;
  wire [2:0]  hps_reset_req;
  wire        hps_cold_reset;
  wire        hps_warm_reset;
  wire        hps_debug_reset;
```

```
  wire [27:0] stm_hw_events;
 // connection of internal logics
assign    stm_hw_events             =    {{3{1'b0}},SW,    fpga_led_internal,
fpga_debounced_buttons};

// ---------------------------- C□o para o PWM -------------------------
// sinais para o clock
  wire [15:0]  largura_de_pulso;    // 16 bits vindo do processador
  reg  [15:0]  dente_de_serra;     // 16 bits para contagem da dente_de_serra
  wire       pwm;                   // 1 bit de PWM para porta GPIO
// Contador da dente_de_serra
    always@(posedge CLOCK_50)
    begin
        if (dente_de_serra >= 50000) begin
        dente_de_serra = 0;
        end else begin
        dente_de_serra = dente_de_serra + 1;
        end
    end
//Comparador do valor da dente_de_serra  com a largura_de_pulso.
    assign pwm = (dente_de_serra > largura_de_pulso)?1'b0:1'b1;
    assign GPIO_0[29] = pwm & 1'b1;//pwm1

// Gera磯 de Valor de Refer□ia por DAC 8 bits
    reg [7:0] Sinal_referencia = 8'd0;

    assign GPIO_0[8] = Sinal_referencia[0]& 1'b1;
    assign GPIO_0[6] = Sinal_referencia[1]& 1'b1;
    assign GPIO_0[4] = Sinal_referencia[2]& 1'b1;
    assign GPIO_0[2] = Sinal_referencia[3]& 1'b1;
    assign GPIO_0[0] = Sinal_referencia[4]& 1'b1;
    assign GPIO_0[1] = Sinal_referencia[5]& 1'b1;
    assign GPIO_0[3] = Sinal_referencia[6]& 1'b1;
    assign GPIO_0[5] = Sinal_referencia[7]& 1'b1;

// ---------------------------- C□o do Sensor de Velocidade Taco-----------
-------

// Definindo Tempo de Contagem
reg [15:0] cont_contagem = 16'd0;
reg        clk_contagem;
reg [32:0] cont_tacho = 32'd0;
reg [32:0] periodo_tacho = 32'd1;   // 32 bits do processador

reg        sinal_aux = 16'd0;
reg        zerar_cont_tacho =16'd0;
// Display
reg [15:0] cont_disp_tacho = 16'd0;
reg           clk_disp_tacho;
reg [15:0] cont_disp_amostra = 16'd0;
reg           clk_disp_amostra;

// Resolu磯 para contagem do periodo do tacho
always@(posedge CLOCK_50) // Divisor de Frequ□ia
begin
        if (cont_contagem >= 2) begin
        clk_contagem = ~ clk_contagem;
        cont_contagem = 0;
        end else begin
        cont_contagem = cont_contagem+1;
        end
```

```
end

//Contagem do periodo do tacho

always@(posedge clk_contagem)
begin
    if (GPIO_0[27] == 0) begin
    // Quando o Sinal estar em nl Baixo
        if (cont_tacho > 60) begin
    // Valor dentro do limite aceit'l de periodo de sinal
        periodo_tacho = cont_tacho;
        cont_tacho = 0;
        end
    // Conta durante o periodo em que o sinal permanece em alto
    end else begin
    cont_tacho = cont_tacho + 1;
    end
end

//interrup磯 de tacho
always@(posedge GPIO_0[27])
begin
    if (cont_disp_tacho >= 220) begin
        cont_disp_tacho = 0;
        clk_disp_tacho = ~ clk_disp_tacho;
    end else begin
        cont_disp_tacho = cont_disp_tacho + 1;
    end
end
//Display referente a saida
assign HEX0[0] = clk_disp_tacho & 1'b1;

    soc_system u0 (

        .signal_pwm_external_connection_export (largura_de_pulso),
          .cont_freq_external_connection_export  (periodo_tacho),
        .sinal_de_referencia_external_connection_export (Sinal_referencia),

        .memory_mem_a                          ( HPS_DDR3_ADDR),
        .memory_mem_ba                         ( HPS_DDR3_BA),
        .memory_mem_ck                         ( HPS_DDR3_CK_P),
        .memory_mem_ck_n                       ( HPS_DDR3_CK_N),
        .memory_mem_cke                        ( HPS_DDR3_CKE),
        .memory_mem_cs_n                       ( HPS_DDR3_CS_N),
        .memory_mem_ras_n                      ( HPS_DDR3_RAS_N),
        .memory_mem_cas_n                      ( HPS_DDR3_CAS_N),
        .memory_mem_we_n                       ( HPS_DDR3_WE_N),
        .memory_mem_reset_n                    ( HPS_DDR3_RESET_N),
        .memory_mem_dq                         ( HPS_DDR3_DQ),
        .memory_mem_dqs                        ( HPS_DDR3_DQS_P),
        .memory_mem_dqs_n                      ( HPS_DDR3_DQS_N),
        .memory_mem_odt                        ( HPS_DDR3_ODT),
        .memory_mem_dm                         ( HPS_DDR3_DM),
        .memory_oct_rzqin                      ( HPS_DDR3_RZQ),

         .hps_0_hps_io_hps_io_emac1_inst_TX_CLK ( HPS_ENET_GTX_CLK),
        .hps_0_hps_io_hps_io_emac1_inst_TXD0   ( HPS_ENET_TX_DATA[0] ),
        .hps_0_hps_io_hps_io_emac1_inst_TXD1   ( HPS_ENET_TX_DATA[1] ),
        .hps_0_hps_io_hps_io_emac1_inst_TXD2   ( HPS_ENET_TX_DATA[2] ),
        .hps_0_hps_io_hps_io_emac1_inst_TXD3   ( HPS_ENET_TX_DATA[3] ),
        .hps_0_hps_io_hps_io_emac1_inst_RXD0   ( HPS_ENET_RX_DATA[0] ),
```

```
.hps_0_hps_io_hps_io_emac1_inst_MDIO   ( HPS_ENET_MDIO ),
.hps_0_hps_io_hps_io_emac1_inst_MDC    ( HPS_ENET_MDC  ),
.hps_0_hps_io_hps_io_emac1_inst_RX_CTL ( HPS_ENET_RX_DV),
.hps_0_hps_io_hps_io_emac1_inst_TX_CTL ( HPS_ENET_TX_EN),
.hps_0_hps_io_hps_io_emac1_inst_RX_CLK ( HPS_ENET_RX_CLK),
.hps_0_hps_io_hps_io_emac1_inst_RXD1   ( HPS_ENET_RX_DATA[1] ),
.hps_0_hps_io_hps_io_emac1_inst_RXD2   ( HPS_ENET_RX_DATA[2] ),
.hps_0_hps_io_hps_io_emac1_inst_RXD3   ( HPS_ENET_RX_DATA[3] ),

  .hps_0_hps_io_hps_io_qspi_inst_IO0     ( HPS_FLASH_DATA[0]    ),
.hps_0_hps_io_hps_io_qspi_inst_IO1     ( HPS_FLASH_DATA[1]    ),
.hps_0_hps_io_hps_io_qspi_inst_IO2     ( HPS_FLASH_DATA[2]    ),
.hps_0_hps_io_hps_io_qspi_inst_IO3     ( HPS_FLASH_DATA[3]    ),
.hps_0_hps_io_hps_io_qspi_inst_SS0     ( HPS_FLASH_NCSO    ),
.hps_0_hps_io_hps_io_qspi_inst_CLK     ( HPS_FLASH_DCLK    ),

  .hps_0_hps_io_hps_io_sdio_inst_CMD     ( HPS_SD_CMD    ),
.hps_0_hps_io_hps_io_sdio_inst_D0      ( HPS_SD_DATA[0]     ),
.hps_0_hps_io_hps_io_sdio_inst_D1      ( HPS_SD_DATA[1]     ),
.hps_0_hps_io_hps_io_sdio_inst_CLK     ( HPS_SD_CLK   ),
.hps_0_hps_io_hps_io_sdio_inst_D2      ( HPS_SD_DATA[2]     ),
.hps_0_hps_io_hps_io_sdio_inst_D3      ( HPS_SD_DATA[3]     ),

  .hps_0_hps_io_hps_io_usb1_inst_D0      ( HPS_USB_DATA[0]    ),
.hps_0_hps_io_hps_io_usb1_inst_D1      ( HPS_USB_DATA[1]    ),
.hps_0_hps_io_hps_io_usb1_inst_D2      ( HPS_USB_DATA[2]    ),
.hps_0_hps_io_hps_io_usb1_inst_D3      ( HPS_USB_DATA[3]    ),
.hps_0_hps_io_hps_io_usb1_inst_D4      ( HPS_USB_DATA[4]    ),
.hps_0_hps_io_hps_io_usb1_inst_D5      ( HPS_USB_DATA[5]    ),
.hps_0_hps_io_hps_io_usb1_inst_D6      ( HPS_USB_DATA[6]    ),
.hps_0_hps_io_hps_io_usb1_inst_D7      ( HPS_USB_DATA[7]    ),
.hps_0_hps_io_hps_io_usb1_inst_CLK     ( HPS_USB_CLKOUT    ),
.hps_0_hps_io_hps_io_usb1_inst_STP     ( HPS_USB_STP    ),
.hps_0_hps_io_hps_io_usb1_inst_DIR     ( HPS_USB_DIR    ),
.hps_0_hps_io_hps_io_usb1_inst_NXT     ( HPS_USB_NXT    ),

  .hps_0_hps_io_hps_io_spim1_inst_CLK    ( HPS_SPIM_CLK  ),
.hps_0_hps_io_hps_io_spim1_inst_MOSI   ( HPS_SPIM_MOSI ),
.hps_0_hps_io_hps_io_spim1_inst_MISO   ( HPS_SPIM_MISO ),
.hps_0_hps_io_hps_io_spim1_inst_SS0    ( HPS_SPIM_SS ),

  .hps_0_hps_io_hps_io_uart0_inst_RX     ( HPS_UART_RX    ),
.hps_0_hps_io_hps_io_uart0_inst_TX     ( HPS_UART_TX    ),

  .hps_0_hps_io_hps_io_i2c0_inst_SDA     ( HPS_I2C1_SDAT    ),
.hps_0_hps_io_hps_io_i2c0_inst_SCL     ( HPS_I2C1_SCLK    ),

  .hps_0_hps_io_hps_io_i2c1_inst_SDA     ( HPS_I2C2_SDAT    ),
.hps_0_hps_io_hps_io_i2c1_inst_SCL     ( HPS_I2C2_SCLK    ),

  .hps_0_hps_io_hps_io_gpio_inst_GPIO09  ( HPS_CONV_USB_N),
.hps_0_hps_io_hps_io_gpio_inst_GPIO35  ( HPS_ENET_INT_N),
.hps_0_hps_io_hps_io_gpio_inst_GPIO40  ( HPS_LTC_GPIO),
//.hps_0_hps_io_hps_io_gpio_inst_GPIO41  ( HPS_GPIO[1]),
.hps_0_hps_io_hps_io_gpio_inst_GPIO48  ( HPS_I2C_CONTROL),
.hps_0_hps_io_hps_io_gpio_inst_GPIO53  ( HPS_LED),
.hps_0_hps_io_hps_io_gpio_inst_GPIO54  ( HPS_KEY),
.hps_0_hps_io_hps_io_gpio_inst_GPIO61  ( HPS_GSENSOR_INT),
  .hps_0_f2h_stm_hw_events_stm_hwevents  (stm_hw_events),
.clk_clk                              (CLOCK_50),
.reset_reset_n                        (hps_fpga_reset_n),
```

```
          .hps_0_h2f_reset_reset_n                (hps_fpga_reset_n),
        .hps_0_f2h_warm_reset_req_reset_n       (~hps_warm_reset),
        .hps_0_f2h_debug_reset_req_reset_n      (~hps_debug_reset),
        .hps_0_f2h_cold_reset_req_reset_n       (~hps_cold_reset)

    );

hps_reset hps_reset_inst (
  .source_clk (CLOCK_50),
  .source     (hps_reset_req)
);

altera_edge_detector pulse_cold_reset (
  .clk       (CLOCK_50),
  .rst_n     (hps_fpga_reset_n),
  .signal_in (hps_reset_req[0]),
  .pulse_out (hps_cold_reset)
);
  defparam pulse_cold_reset.PULSE_EXT = 6;
  defparam pulse_cold_reset.EDGE_TYPE = 1;
  defparam pulse_cold_reset.IGNORE_RST_WHILE_BUSY = 1;

altera_edge_detector pulse_warm_reset (
  .clk       (CLOCK_50),
  .rst_n     (hps_fpga_reset_n),
  .signal_in (hps_reset_req[1]),
  .pulse_out (hps_warm_reset)
);
  defparam pulse_warm_reset.PULSE_EXT = 2;
  defparam pulse_warm_reset.EDGE_TYPE = 1;
  defparam pulse_warm_reset.IGNORE_RST_WHILE_BUSY = 1;

altera_edge_detector pulse_debug_reset (
  .clk       (CLOCK_50),
  .rst_n     (hps_fpga_reset_n),
  .signal_in (hps_reset_req[2]),
  .pulse_out (hps_debug_reset)
);
  defparam pulse_debug_reset.PULSE_EXT = 32;
  defparam pulse_debug_reset.EDGE_TYPE = 1;
  defparam pulse_debug_reset.IGNORE_RST_WHILE_BUSY = 1;


endmodule
```