UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E INFORMÁTICA INDUSTRIAL

ALISSON ANTÔNIO DE OLIVEIRA

ESTUDO E IMPLEMENTAÇÃO DE OPERAÇÕES EM PONTO FIXO EM FPGA COM VHDL 2008: APLICAÇÃO EM CONTROLE DE SISTEMAS EM TEMPO DISCRETO

DISSERTAÇÃO

CURITIBA
2012

ALISSON ANTÔNIO DE OLIVEIRA

ESTUDO E IMPLEMENTAÇÃO DE OPERAÇÕES EM PONTO FIXO EM FPGA COM VHDL 2008: APLICAÇÃO EM CONTROLE DE SISTEMAS EM TEMPO DISCRETO

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do grau de "Mestre em Ciências" – Área de concentração: Engenharia de Automação e Sistemas.

Orientação: Prof. Dr. Carlos Raimundo Erig Lima

CURITIBA
2012

Dados Internacionais de Catalogação na Publicação

O48 Oliveira, Alisson Antônio de
Estudo e implementação de operações em ponto fixo em FPGA com VHDL 2008: aplicação em controle de sistemas em tempo discreto / Alisson Antônio de Oliveira. – 2012.
137 f. : il. ; 30 cm

Orientador: Carlos Raimundo Erig Lima.
Dissertação (Mestrado) – Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial. Curitiba, 2012.
Bibliografia: f. 100-104.

1. Sistemas de controle digital. 2. VHDL (Linguagem descritiva de hardware). 3. Arranjos de lógica programável em campo. 4. Cálculos numéricos. 5. Controle em tempo real. 6. Simulação (Computadores). 7. Engenharia elétrica – Dissertações. I. Lima, Carlos Raimundo Erig, orient. II. Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial. III. Título.

CDD (22. ed.) 621.3

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Câmpus Curitiba

Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial

Titulo da Dissertação Nº. 621

"Estudo e Implementação de Operações em Ponto Fixo em FPGA com VHDL 2008: Aplicação em Controle de Sistemas em Tempo Discreto"

por

Alisson Antônio de Oliveira

Esta dissertação foi apresentada como requisito parcial à obtenção do grau de MESTRE EM CIÊNCIAS – Área de Concentração: Engenharia de Automação e Sistemas, pelo Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial – CPGEI – da Universidade Tecnológica Federal do Paraná – UTFPR, às 09h do dia 13 de dezembro de 2012. O trabalho foi aprovado pela Banca Examinadora, composta pelos professores:

Prof. Carlos Raimundo Erig Lima, Dr.
(Presidente – UTFPR)

Prof. Andre Schneider de Oliveira, Dr.
(IFPR)

Prof. João Alberto Fabro, Dr.
(UTFPR)

Visto da coordenação:

Prof. Ricardo Lüders, Dr.
(Coordenador do CPGEI)

A Folha de Aprovação assinada encontra-se na coordenação do programa

Dedico essa dissertação aos meus pais, que me educaram e ensinaram os valores da vida; aos meus irmãos que foram meus primeiros amigos e nunca deixarão de ser; e a minha querida esposa que soube compreender a minha ausência durante essa etapa importante da vida.

AGRADECIMENTOS

Agradeço a UTFPR pelo suporte desde a graduação até esse momento.

Agradeço ao professor Carlos Erig pelas piadas incentivadoras.

Aos colegas do laboratório de Inovações Tecnológicas (LIT) que encontrei nesses anos e que me ajudaram em dúvidas técnicas.

Aos professores membros da banca, por aceitarem o convite e pelas preciosas contribuições no meu trabalho.

Finalmente, agradeço a todos que de alguma forma, contribuíram direta ou indiretamente nesse trabalho para que o mesmo pudesse ser concluído.

Não existe vento favorável para o marinheiro que não sabe aonde ir. (Sêneca).

Como dizem os construtores, as pedras maiores não ficariam bem assentadas sem as menores. (Platão 428-348 A.C.).

RESUMO

OLIVEIRA, Alisson Antônio de. **Estudo e implementação de operações em ponto fixo em FPGA com VHDL 2008: aplicação em controle de sistemas em tempo discreto.** 137 p. Dissertação (Mestrado em Engenharia Elétrica e Informática Industrial) – Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI), Universidade Tecnológica Federal do Paraná (UTFPR), Curitiba, 2012.

Existem máquinas que necessitam de uma grande velocidade de processamento para seu correto trabalho, essas máquinas possuem um tempo de processamento de resposta crítico. Quando considera-se este aspecto somado à necessidade de um controle do comportamento estático e dinâmico de um sistema chega-se ao controlador com fortes demandas de tempo de execução. Essa dissertação compara controladores discretos implementados em ponto fixo, com diferentes precisões, usando para tanto a simulação do comportamento de controladores confeccionados em linguagem de comandos Matlab e em linguagem VHDL 2008. Esta última está em desenvolvimento e padronização pelo IEEE. A linguagem VHDL é usada nas FPGAs que são dispositivos de alta velocidade e capacidade de processamento paralelo. O principal objetivo do trabalho é o estudo e a implementação de controladores discretos em FPGA com o auxílio da linguagem VHDL 2008, determinando suas virtudes e limitações, em particular quanto à estrutura de programação, análise de erro e a demanda por recursos. Os resultados alcançados demonstram que algumas melhorias ainda precisam ser feitas para que o VHDL 4.0, conhecido como VHDL 2008, seja entregue ao mercado como padrão estável. Entretanto, quando conhecidas suas limitações, já é possível seu uso em implementações com conversão de sinais discretos para analógicos, como é o caso de controle e simulação de sistemas dinâmicos como servomecanismos.

Palavras-chave: Controle digital. VHDL 2008. Cálculo numérico. Controle em tempo real. Simulação em hardware reconfigurável.

ABSTRACT

OLIVEIRA, Alisson Antônio de. **Study and implementation of operations in fixed point with FPGA VHDL 2008: Application on discrete time control systems**. 137 p. Dissertação (Mestrado em Engenharia Elétrica e Informática Industrial) – Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI), Universidade Tecnológica Federal do Paraná (UTFPR), Curitiba, 2012.

There are machines that need large processing speed for its correct working, these machines have a critical time response processing. When it is considered that aspect coupled with the need for control of static and dynamic behavior of a system arrives at the controller with strong demands on runtime. This dissertation compares discrete controllers implemented in fixed point with different accuracies, using for both the simulation of the behavior of controllers manufactured in Matlab command language and VHDL 2008. VHDL 2008 still in development and standardization by the IEEE. The VHDL language is used in FPGAs that are high speed devices with parallel processing capability. The main objective of this work is the study and implementation of discrete controllers in FPGA with the help of the VHDL 2008 language, determining its strengths and limitations, particularly in regard to the structure of programming, error analysis and demand for resources. Results show that accuracy still need some improvements a standard to the VHDL 4.0, known as VHDL 2008, is delivered to the market a stable standard. However, knowing it limitations, it is possible implementations and use in conversion of analog signals to discrete, such as control and dynamic systems simulation like servomechanisms.

Keywords: Discrete control. VHDL 2008. Numerical calculation. Real-time control. Simulation on reconfigurable hardware.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

AC	Corrente alternada (*alternate current*)
AD	Analógico para digital (conversão)
APSO	Otimização por enxame de partículas de forma assíncrona (*asynchronous particle swarm optimization*)
ASCII	Código americano padrão para troca de mensagens (*American Standard Code for Information Interchange*)
CLB	Blocos Lógicos Configuráveis (*Configurable Logic Blocks*)
CPLDS	Dispositivos lógicos programáveis complexos (*Complex programmable logic device*)
CNC	Comando numérico computadorizado
DA	Digital para analógico (conversão)
DC	Corrente contínua (*direct current*)
DSP	Processador digital de sinais (*Digital Signal Processor*)
EDA	Projeto eletrônico automatizado (*Electronic Design Automation*)
EKF	Filtro estendido de Kalman (*extended Kalman filter*)
FAQ	Questões frequentemente perguntadas (*Frequently Asked Questions*)
FPGA	(F*ield-Programmable Gate Array*)
GPI	Controle proporcional integral generalizado (*generalized proportional integral*)
GUI	Graphical user interface (interface gráfica de usuário)
HDL	Linguagem de descrição de hardware (*hardware description language*)
HIL	Simulação em hardware de sistema de controle (*Hardware-in-the-loop*)
IEEE	Instituto dos engenheiros elétricos e eletrônicos (*Institute of Electrical and Electronics Engineers*)
IP	Propriedade Intelectual (*Intellectual property*)
IIR	Resposta infinita ao impulso (*Infinite impulse response*)
LABS	Matriz de blocos lógicos (*Logic Array Blocks*)
LUT	Estrutura de dados de forma matricial (*Lookup Table*)
MOSFET	(*Metal Oxide Semiconductor Field Effect Transistor*)
NURBS	(N*on-Uniform Rational B-Spline*)
PC	Computador pessoal (*Personal Computer*)
PID	Proporcional, Integral e Derivativo (arquitetura de controlador)
PLD	(P*rogrammable Logic Device*)
PSO	Otimização por enxame de partículas (*Particle Swarm optimization*)
PWM	Modulação por largura de pulso (*Pulse-Width Modulation*)
RAM	Memória de acesso aleatório (*Random Access Memory*)
SoC	Sistema em um Chip (*System-on-a-chip*)
TCL	Linguagem de Linha de comando (*Tool Command language*)
VHDL	Linguagem de descrição de hardware VHSIC (VHSIC *Hardware Description Language*)
VHSIC	Circuitos integrados de alta velocidade (*Very High Speed Integrated Circuits*)
VLSI	Integração em Larga Escala (*Very Large Scale Integration*)

SUMÁRIO

1. INTRODUÇÃO

O objetivo desse capítulo é fornecer uma introdução sobre os principais itens que compõem a dissertação, motivações, objetivos e a forma de organização dessa dissertação.

Neste capítulo, a seção 1.1 descreve as motivações encontradas para a pesquisa que resultou nesse trabalho. A seção 1.2 apresenta os objetivos pretendidos. Na seção 1.3 comenta-se sobre a metodologia do trabalho e, por fim, a seção 1.4 apresentada a organização da dissertação.

1.1 MOTIVAÇÃO

O avanço tecnológico permitiu a criação de equipamentos eletromecânicos rápidos demandando métodos e técnicas de controle capazes de atender estas demandas dinâmicas e de desempenho. Por exemplo, os servomotores usados em máquinas ferramentas ou mesmo os motores *brushless* usados em unidades de armazenamento de dados demandam controles sofisticados e com requisitos de tempo de execução críticos.

Uma abordagem computacional, baseada em dispositivos lógicos reconfiguráveis, a exemplo das FPGA, pode oferecer uma alternativa para as arquiteturas convencionais baseadas em execução de software. De acordo com Ito e Carro (2000), algumas das vantagens da reconfiguração por hardware são:

- Paralelismo real, em oposição ao modelo de Von Newmann.
- Desenvolvimento de projeto modular e hierárquico.
- Redução do tempo de projeto, permitindo metodologias de projeto *top-down* e *bottom-up*.
- Inúmeras plataformas de desenvolvimento.
- Estão à disposição grande número de funções testadas (IP-core), reduzindo o ciclo de projeto com funções muito complexas.

Além disto, segundo Józmiak, Nedjah e Figueroa (2010), há numerosos exemplos de projetos usando sistemas reconfiguráveis por *hardware*, com maior desempenho, menor consumo, menor custo e menor tempo de desenvolvimento que as soluções equivalentes baseadas em processadores convencionais.

A busca por sistemas que possam controlar equipamentos de forma simples, com precisão e rapidez trás como possibilidade o uso de FPGAs na área de controle de sistemas em tempo discreto, também chamado de controle digital. A organização de normas *Accellera Systems Initiative* (associado ao *Institute of Electrical and Electronic Engineers* - IEEE) e a EDA *Industry Working Groups* organizaram, no início de 2008, o VHDL 4.0, também conhecido como VHDL 2008. Nesta versão foi apresentado um pacote com definições de operações matemáticas com vírgula fixa e vírgula flutuante (também chamadas de ponto fixo e ponto flutuante devido ao termo em inglês) para operações em VHDL (*VHSIC Hardware Description Language*) (BISHOP, 2012).

Estes novos recursos matemáticos do VHDL 2008 permitiram a construção de controladores discretos mais precisos que os controladores construídos com representação por números inteiros. Por outro lado, abre-se a possibilidade de duas abordagens: a utilização do ponto fixo ou a utilização do ponto flutuante. Neste trabalho, optou-se pela análise de diferentes implementações em ponto fixo. Esta escolha se deve a dois fatores:

- A menor demanda por recursos da FPGA para implementação com ponto fixo.
- Nova forma de interfaceamento dos dados a serem analisados uma vez que comandos expecíficos e simples foram criados.

Com base nesses itens percebesse uma demanda pelo estudo de controladores clássicos em FPGAs usando os novos recursos matemáticos presentes no VHDL 2008.

1.2 OBJETIVOS

O objetivo desse trabalho é o estudo e a implementação de controladores discretos em FPGA com a linguagem VHDL 2008, determinando suas virtudes e limitações, em particular quanto à estrutura de programação, análise de erro e a demanda por recursos para a área de controle de sistemas.

Para realização deste objetivo, por motivo de larga utilização na indústria, serão utilizados controladores PID (Proporcional, Integral e Derivativo) discretos e

alguns modelos de plantas descritas na literatura e comumente controladas por este tipo de controlador.

Em consonância com o objetivo proposto, alguns objetivos secundários são apresentados:

- Construção de uma ferramenta de simulação em Matlab para realização dos mesmos cálculos em ponto fixo posteriormente realizados em *hardware*.
- Estudo da demanda por recursos da FPGA para cada tipo de controlador, modelo de planta e sua respectiva precisão;
- Estudo do impacto de desempenho e demanda por recursos de sistemas discretos (plantas + controladores) com diferentes comportamentos físicos.
- Validação dos controladores e plantas discretas implementadas através da comparação da resposta dinâmica dos mesmos com simuladores tradicionais em *software*. A validação em sistemas reais não será realizada, pois isso depende de recurso e tempo para a construção desses modelos físicos reais.
- Análise numérica do erro em função de diferentes opções de precisão.

Após a descrição dos objetivos do trabalho se faz necessário apresentar a metodologia que será abordada para a execução dos mesmos.

1.3 METODOLOGIA

Todo o processo usado nessa dissertação pode ser dividido basicamente em três partes, são elas:

- Definição e estudo das plantas e controlador a serem utilizados: nesse momento o estudo recai sobre a taxa de amostragem, o tipo de controlador e as características dinâmicas da planta;
- Implementação das plantas e controlador em Matlab: nesse ponto todo o sistema está discretizado e a quantidade de bits representativos para as constantes dos controladores, bem como operações matemáticas e valores guardados na memória para uso na amostragem seguinte são alterados gradativamente para analisar o comportamento do sistema;
- Implementação das plantas e controlador em *hardware*: nesse último passo o sistema é implementado na linguagem VHDL 2008 e duas informações

importantes são coletadas: a quantidade de unidades lógicas utilizadas e o comportamento dinâmico e estático do sistema.

1.4 ORGANIZAÇÃO DA DISSERTAÇÃO

Esta dissertação está organizada em seis capítulos. O capítulo 2 descreve os controladores discretizados e suas características básicas, focando nos controladores PID. Também será apresentado um resumo sobre a linguagem VHDL usada nas FPGAs e logo depois uma descrição dos aspectos principais sobre a utilização do ponto fixo na linguagem VHDL 2008. Será apresentada também uma descrição sobre erros em cálculo numérico e limites de precisão, necessários para avaliação da adequação dos resultados numéricos obtidos.

O capítulo 3 fala sobre o estado da arte. Ele apresenta uma pesquisa sobre os artigos em congressos e revistas que falam sobre o conjunto FPGA e controle de sistemas discretos.

O capítulo 4 fala sobre a ferramenta desenvolvida para trabalhar com a linguagem VHDL 2008 que está em processo de desenvolvimento. O capítulo 5 fala sobre os experimentos propostos, os resultados alcançados e as conclusões observadas em cada experimento. O capítulo 6 apresenta a conclusão geral do trabalho, as dificuldades e méritos encontrados e propõe trabalhos futuros.

Finalmente foram criados, no final da dissertação, os apêndices com gráficos e tabelas dos testes realizados que são comentados ao longo da dissertação.

2. FUNDAMENTAÇÃO TEÓRICA

Nesse capítulo será feita a descrição dos elementos fundamentais para compreensão dos estudos realizados na dissertação. O capítulo vai tratar de controladores digitais, FPGAs, VHDL 2008, métricas de controle e erros provenientes de cálculos numéricos com limite de precisão.

2.1 CONTROLADORES DISCRETOS

Os controladores discretos tem seu desempenho limitado pela aritmética digital e pela taxa de amostragem utilizada. Assim, por exemplo, um controlador que opera com aritmética de inteiros e taxa de amostragem de 100ms, pode ser adequado para um controle de posição de um motor de grande porte. Tipicamente as operações matemáticas realizadas entre cada amostragem são executadas com folga pelo hardware associado. Por outro lado, este mesmo controlador pode ser absolutamente inadequado para controlar a posição de um motor menor e com menor carga. Tal inadequação pode se dar pela falta de precisão das operações de aritmética digital e/ou capacidade de processamento entre cada amostragem.

Um diagrama básico de controle é apresentado na figura 1. Mediante a definição de um sinal de referência e da obtenção do sinal de realimentação negativa é gerado o sinal de erro. Em função do sinal de erro, o controlador gerará um sinal de controle para o atuador. O objetivo é levar a planta ao valor de referência desejado, segundo alguma restrição dinâmica, como por exemplo sobre sinal máximo ou tempo de subida máximo.

Por exemplo, no caso de um forno industrial, a referência poderia ser de 200ºC, o controlador poderia ser do tipo PID, o atuador poderia ser uma resistência elétrica que aquece o forno e a planta é o forno propriamente dito, operando na temperatura desejada.

Do ponto de vista de controle a falta de precisão pode gerar erro no comportamento final do processo, um exemplo é indicado no item 2.2.1 desta dissertação.

Um diagrama em blocos da área de controle é indicado na figura 1, ele representa a arquitetura clássica usada no controle de sistemas.

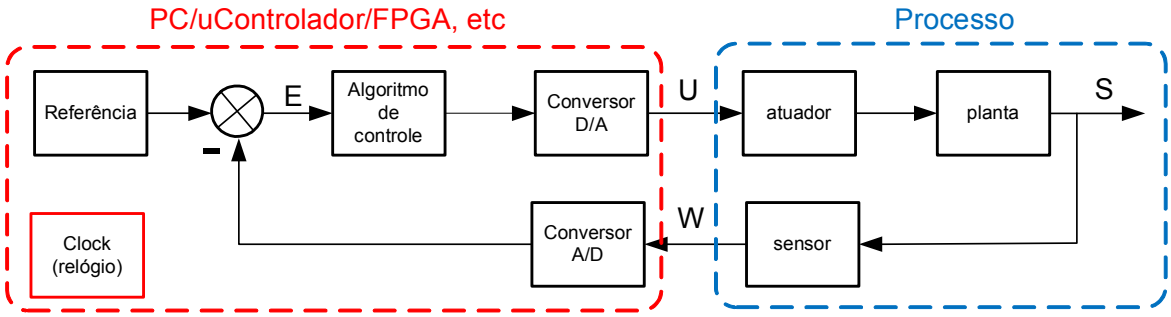


Figura 1 – Esquema básico de controle discreto
Fonte: Adaptado de Astrom e Wittenmark (1997) p. 2 e Dorf e Bishop (2001).

Os sinais E, U, W e S são:

- E - Sinal de erro: é a diferença entre a referência desejada e o sinal adquirido pelo conversor analógico-digital (A/D);
- U - Sinal de controle: esse é o sinal que o controlador, através do conversor digital-analógico (D/A), envia para o atuador após cálculos que modificam as características dinâmicas e estáticas do sistema;
- W - Sinal do sensor: esse é o sinal da leitura do sensor associado a variável a ser controlada na planta;
- S - Sinal de saída da planta: esse é o estado atual da planta.

Segundo Astrom e Wittenmark (1997) os sistemas de controle modernos são implementados quase que exclusivamente em controle baseado em processadores ou microprocessadores e existem alguns fenômenos que acontecem nesses sistemas discretos que não são observados nos sistemas analógicos, sendo fundamental para os engenheiros de controle o correto entendimento destes fenômenos.

Um exemplo clássico são os conversores AD e DA trabalhando a uma taxa de amostragem pré-fixada. Estes conversores introduzem distorções na operação do sistema que devem ser consideradas pelo engenheiro de controle.

Ainda segundo Astrom e Wittenmark (1997) o uso de sistemas computadorizados em controle de sistemas dinâmicos se iniciou em meados de 1950, com aplicações iniciais em controle de mísseis e veículos aéreos (*aircrafts*). Devido a aplicações específicas de alto desempenho e complexidade foi necessário o desenvolvimento de equipamentos de propósito especial para a solução de problemas técnicos de processamento, a exemplo dos analisadores diferenciais digitais (*digital differential analyzers* - DDA).

Para a otimização dos controladores é necessário o uso de modelos matemáticos discretos que representam o comportamento do sistema. Assim, conhecendo-se de forma relativamente precisa o modelo matemático discreto da planta é possível projetar controladores discretos otimizados. As características marcantes dos controles digitais são:

- Flexibilidade: os controladores analógicos possuem várias limitações físicas associadas aos componentes, como resistores, capacitores, entre outros;
- Facilidade na interação com vários laços de repetição (*loops*) de controle: os sistemas discretos possuem flexibilidade devido à facilidade de reprogramação;
- Os parâmetros de controle podem ser criados em função das condições de operação, ou seja, os parâmetros podem variar mediante regras bem definidas.

Ainda com Astrom e Wittenmark (1997, p.10), com base em observações existem quatro áreas que vão ser fundamentais para o desenvolvimento do controle discreto, são elas:

- Conhecimento sobre o processo a ser controlado;
- Tecnologia de medição e sensoriamento;
- Tecnologia dos computadores e seus semelhantes;
- Teoria de controle.

O primeiro item apresenta o conhecimento prévio do comportamento da planta e como coletar os dados da planta a ser controlada mediante a utilização de sensores. Entretanto é necessária uma análise para verificar as possíveis dificuldades de controle de cada planta, a exemplo de não linearidades e variabilidade no tempo.

As tecnologias de medição e sensoriamento estão diretamente ligadas ao desenvolvimento tecnológico de novos materiais. Além disso, técnicas de fabricação e a capacidade de processamento também sofrem melhorias, permitindo sensores mais precisos e com menor custo de produção, bem como a execução de algoritmos mais complexos.

Com relação às novas teorias que envolvem o controle de sistemas dinâmicos, o controle preditivo baseado em modelos de referência internos e o

controle adaptativo são as técnicas de maior destaque na indústria (Astrom e Wittenmark, 1997, p. 11).

2.1.1 Amostrando sinais contínuos

A amostragem é uma propriedade fundamental no controle de sistemas discretizados. Ela consiste na aquisição de dados de uma planta e posterior cálculo pelo controlador, segundo uma taxa de tempo fixa. A princípio, esta taxa de aquisição não deve variar, podendo implicar numa resposta do controlador diferente da especificada ou esperada. Existem casos específicos de controladores que usam amostragens com várias taxas, mas são controladores específicos que não serão abordados nesse trabalho.

A amostragem é feita por um conversor de dados analógicos para digitais (A/D) com uma frequência fixa de operação. Caso seja necessário reconstruir o sinal de saída é usado o conversor digital para analógico (D/A), conforme indicado na figura 1. Tocci e Widmer (2007) comentam sobre outras técnicas de amsotragem e interfaceamento, como por exemplo, a conversão de analógico para digital pela técnica sigma/delta.

Um estudo sobre as taxas de amostragem é apresentado no apêndice A, cuja principal conclusão é: a taxa de amostragem é fundamental para descrever a real dinâmica de um sistema a ser controlado. Caso a amostragem não seja corretamente estimada, o controlador pode levar ao funcionamento inadequado ou mesmo a não estabilidade do sistema. Quanto maior é a taxa de amostragem melhor é a visualização da dinâmica da planta.

2.1.2 Analisando um sistema discreto

Existem elementos na área de controle que são fundamentais para análise da boa qualidade do controlador desenvolvido. Segundo Astrom e Wittenmark (1997, p. 77) dentro da área de controle discreto elas são:

- Estabilidade: a definição mais simples de estabilidade de um sistema é que a saída é limitada para uma entrada limitada. Existem outras definições em Astrom e Wittenmark (1997), mas a indicada acima é a mais simples. Para

testar a estabilidade existem métodos conhecidos, como por exemplo: Critério de Jury, Nyquist, Lyapunov e alocação de pólos;

- Sensibilidade: esse item descreve a sensibilidade do sistema com perturbações externas sobre o controle e planta;
- Robustez: a robustez é o item que descreve o quão preparado está o controlador projetado para operar sobre uma planta “real” e não ideal, pois a planta real possui pequenas variações entre o projetado (simulado) e o verdadeiro. Quanto mais robusto for o controlador, mais variações (erros) da planta ele aceita, entretanto quanto mais robusto for o controlador menor é seu desempenho;
- Controlabilidade: a teoria da controlabilidade diz que o sistema é controlável se for possível achar uma sequência de controle finita que possa levar o sistema a qualquer posição do espaço de estados em um tempo finito;
- Observabilidade: um sistema é dito observável no tempo t_o se, com o sistema no estado $x(t_o)$, for possível determinar este estado apenas observando-se a saída (e a entrada) do sistema durante um intervalo de tempo finito.

Existem outros conceitos como acessibilidade e detectividade (Astrom e Wittenmark, 1997, p. 93), entretanto eles são menos conhecidos e não serão comentados nesse trabalho.

Os motivos do uso de realimentação nos sistemas de controle são (Astrom e Wittenmark, 1997, p. 103):

- Melhorar a resposta transitória do sistema;
- Diminuir a sensibilidade à alteração dos parâmetros em malha aberta;
- Eliminar o erro em estado estacionário caso a planta já não possua essa característica;
- Diminuir a influência das perturbações provenientes de cargas e problemas com erros nas medidas;

Dentro do conjunto de problemas que podem acontecer dentro do controle, existem alguns que o controlador deve sobrepujar. Esses problemas podem ser divididos da seguinte forma:

- Perturbações de carga: a variação na carga acaba influenciando as variáveis do processo. Ela pode aparecer de várias formas e isso depende do processo e atuador. Por exemplo, para um motor seria a carga acoplada ao seu eixo,

para um navio seriam as ondas, para uma antena seriam os ventos e para uma geladeira seria a variação da temperatura ao abrir a porta;

- Erros de medida: esses erros são encontrados frequentemente nos sensores. Eles podem acontecer pela não calibração adequada no sensor, originando um erro em estado estacionário. Entretanto é possível visualizar pequenas variações em alta frequência que são conhecidos como ruídos. Esses ruídos podem ser amenizados com filtros colocados entre os sensores e os controladores;
- Variação de parâmetros: o controle clássico se baseia na representação de sistemas lineares. Na prática os sistemas reais são, via de regra, não lineares (OGATA, 1998). Dessa forma, o controlador deve controlar o sistema satisfatoriamente em um determinado regime de operação, apesar das não linearidades.

Existem alguns tipos de modelos de distúrbios simples que podem ser usados durante o projeto dos controladores. São eles: impulso (*pulse*), degrau (*step*), rampa (*ramp*) e sinal senoidal (*sinusoid*), descritos na figura 2.

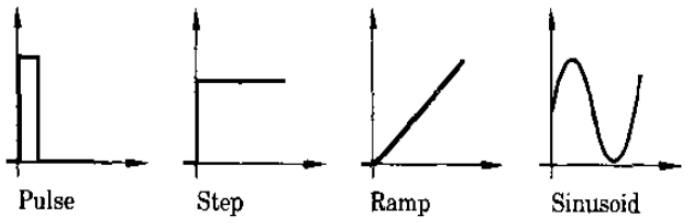


Figura 2 – Exemplos de referências usadas em controle
Fonte: Astrom e Wittenmark (1997, p. 104).

Astrom e Wittenmark (1997) comentam sobre a grande importância que a simulação possui sobre o projeto de controladores, mas advertem que a simulação pode não ser suficiente. É necessário analisar constantemente os resultados para observar se os controladores são aplicáveis à planta. Através da simulação não é possível investigar todas as combinações de perturbações existentes. Os autores ainda relatam alguns métodos de controle não encontrados em Ogata (1998) e Dorf e Bishop (2001), como por exemplo, o controlador DeadBeat, Preditor Smith e Controle por Modelo Interno (IMC). Entretanto como eles são baseados em operações de matrizes e não em equações a diferenças, não serão abordados nesse trabalho.

De forma simplificada, as metodologias para se projetar e estruturar os controladores são (Astrom e Wittenmark, 1997, p. 229):

- *Top-down*: a pesquisa se inicia com a definição do problema e a cada nova etapa são acrescentados mais detalhes até que todo o problema esteja bem conhecido.
- *Bottom-up*: todas as pequenas “peças” conhecidas que representam um subproblema são agrupadas, elas são combinadas até que a solução do problema seja obtida.

O propósito do controle é manter o sistema operando próximo de um valor estimado sem que distúrbios ou variações do processo atrapalhem esse processo (DORF e BISHOP, 2001). Nesse aspecto existe a possibilidade da especificação pelo controle de variância mínima ou controle otimizado. Na primeira, uma distribuição Gaussiana é usada para saber se os produtos estão corretos, mas sem variação excessiva do sinal de controle. No segundo o sinal de controle não é poupado e pode variar rapidamente em busca da otimização do sistema.

Para resumir, os itens básicos necessários para análise no momento do projeto de controladores podem ser representados por:

- Sinal de comando;
- Distúrbios da carga;
- Ruído de sinal;
- Incertezas do modelo;
- Saturação do atuador;
- Restrições de estado;
- Complexidade do controlador.

No subitem seguinte será comentado sobre as arquiteturas de controle encontradas na literatura.

2.2 ARQUITETURAS BÁSICAS DE CONTROLADORES

Existem arquiteturas de controladores consideradas clássicas devido ao tempo em que já são conhecidas e utilizadas tanto na academia como na indústria.

Dentre essas arquiteturas, a mais usada é o PID. PID é a sigla para controlador Proporcional, Integral e Derivativo.

De forma simplificada, as famílias dos controladores clássicos serão agrupadas em subitens, apresentados como (OGATA, 1998):

- Controlador de duas posições ou liga-desliga (on-off);
- Controlador proporcional;
- Controlador proporcional e integral;
- Controlador proporcional e derivativo;
- Controlador proporcional, integral e derivativo;
- Controlador por avanço de fase;
- Compensador por atraso de fase;
- Controlador por avanço e atraso de fase.

Para sistemas de controle mais modernos não definidos como SISO (*single input single output*) é recomendado o controle por espaço de estados. O mesmo baseia-se na representação matricial de sistemas com múltiplas variáveis e/ou múltiplas entradas e saídas, ou seja, sistemas MIMO (*multiple input multiple output*).

A complexidade da implementação das operações matriciais para representação e controle de sistemas em FPGA demandaria um esforço que poderia ser tratado como outra dissertação de mestrado, não sendo contemplado neste trabalho.

2.2.1 Família PID

O controlador PID é um dos mais, se não o mais, comum controlador usado na indústria (ASTROM, HUGGLAND, 1995). Existem algumas pequenas variações em suas arquiteturas. Este controlador é tão importante que existem vários autores que escreveram livros exclusivos sobre o mesmo. Como referência pode-se citar: Astrom e Hagglund (1995), Astrom e Hagglund (2006), Choi e Chung (2004), Datta, Ho e Battacharyya (2000a), Datta, Ho e Battacharyya (2000b), Johnson e Moradi (2005), O´Dwyer (2006), Quevedo e Scobet (2000), Silva, Datta e Battacharyya (2005), Vilanova e Visioli (2012), Visioli (2006), Wang, Ye, Cai e Hang (2008), Yu (1998).

Além disto, de forma genérica, quando se fala em controle de sistemas dinâmicos o primeiro controlador a ser levando em consideração como proposta de solução técnica é o controlador PID (TEIXEIRA, 2012).

A existência ou ausência das ações de controle proporcional, integral e derivativa permitem a definição de subconjuntos do controlador PID. O primeiro elemento de estudo dos controladores PID é a ação proporcional, representada pelo controle apenas proporcional (P), indicado aqui como Kp. Ele trabalha com a multiplicação do índice de erro para gerar o sinal de controle, representado matematicamente pela equação 1.

$$u = e * Kp \tag{1}$$

O controlador PI possui um elemento muito importante que é o elemento de integração, usado tipicamente para zerar o erro em regime permanente. O elemento integral pode desestabilizar a planta se for mal projetado e está diretamente relacionado com o tempo de amostragem em sistemas discretos. A equação 2, apresenta o controlador PI (K+I).

$$u = e * Kp + \frac{Kp}{Ti}\int_0^t e * dt \tag{2}$$

Onde, o termo Kp/Ti é usualmente representado por Ki.

Como todos os sistemas físicos possuem limitações, essas limitações podem trazer problemas para o controlador PI, pois são representadas como não linearidades do sistema. No caso de uma ação com saturação, o ganho integral pode ir acumulando um erro e gerando um sinal de controle acima do máximo que o atuador pode suportar. Para que isso não aconteça, existe a possibilidade da implantação de um integrador anti-windup que nada mais é que a determinação de um limite máximo (positivo e negativo) para os possíveis valores do integrador, limitando o acúmulo de erro integral que pode saturar o atuador da planta.

O controlador proporcional derivativo (PD) possui o ganho proporcional mais o ganho derivativo. O elemento derivativo trabalha com a taxa de variação do sinal de erro do sistema. Uma vez que ele se baseia em uma derivada, quando a planta entra em regime permanente este elemento tem uma contribuição igual à zero, pois a derivada de uma constante é igual à zero. É importante comentar que em plantas com muito ruído no sinal do sensor, o sistema pode ter sérias oscilações quando um ganho derivativo é mal projetado. Isto ocorre pelo fato do controlador PD amplificar

de forma indevida altas frequências, onde geralmente estão presentes os ruídos. Como solução pode-se utilizar um controlador PD modificado, adicionando-se um pólo de alta frequência ao mesmo. A arquitetura do controlador PD é apresentada na equação 3.

$$u = e * Kp + Kp * Td * \frac{de}{dt} \quad (3)$$

onde, o termo Kp*Td é chamado comumente de Kd.

O controlador PID então engloba os três elementos em uma única equação, agregando as três ações de controle, multiplicação por uma constante, soma do erro e ação sobre as variações no sinal de erro.

A equação completa do PID (P+I+D) é apresentada na equação 4.

$$u = e * Kp + Ki \int_0^t e * dt + Kd * \frac{de}{dt} \quad (4)$$

A representação do diagrama de blocos do controlador PID pode ser visualizada na figura 3.

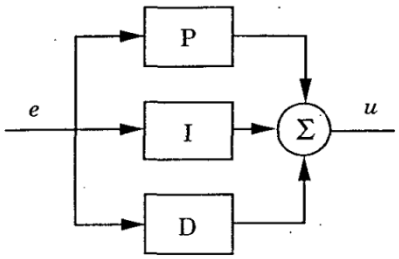


Figura 3 – Diagrama de blocos do controlador PID paralelo
Fonte: Astrom e Hagglund (1995).

Os controladores PID foram inicialmente implementados em sistemas analógicos, mas com a queda nos preços dos microcontroladores e microprocessadores a relação custo benefício tendeu para sua implementação de forma digital, pois os sistemas digitais são menos sensíveis a variações do ambiente como temperatura, pressão e umidade.

Quando um sistema discreto é implementado para o cálculo dos controladores digitais é interessante gerar seu algoritmo conforme a sequência abaixo (ASTROM, HAGGLUND, 1995, p. 93):

1. Espere pelo sinal de clock (amostragem).
2. Leia a entrada analógica e converta para discreta.
3. Calcule o sinal de controle mediante função de controle.
4. Envie o sinal de controle para a saída.
5. Faça a atualização (*update*) das variáveis internas de controle.
6. Volte para o passo 1.

Para as conversões de equações no domínio do tempo contínuo para o tempo discreto cada uma das formas de conversão será vista no subitem 2.5.1 que fala exclusivamente sobre os métodos numéricos.

Com relação à quantização e a limitação da representatividade dos números dentro de sistema discreto, Astrom e Hugglund (1995) comentam sobre o truncamento do sinal de saída. Com base nessa falta de precisão numérica e por consequência uma possível geração de erro em regime permanente é necessária a implementação do "bias". O bias é uma constante somada ao sinal de controle final para zerar (anular) o erro em regime permanente ocasionado pela falta de precisão numérica.

Como exemplo da necessidade de precisão pode-se fazer a seguinte conta: tendo um ganho proporciona igual a 0,002 e um tempo de integração de 20 min (1200 s) é possível achar seu valor final (decimal) e quantos bits são necessários para a sua representação. A equação 5 apresenta esses cálculos.

$$\frac{Kp}{Ti} = \frac{0{,}002}{1200} = 1{,}666 * 10^{-6} \rightarrow 2^{-19{,}2} \quad (5)$$

Para a correta representação são necessários 20 bits, muito embora valores próximos possam apresentar margens aceitáveis de erro.

Astrom e Hagglund (1995) apresentam uma tabela comparativa entre vários fabricantes de ferramentas que usam os controladores PID. Essa lista de empresas é apresentada abaixo (Astrom e Hagglund, 1995, p. 110):

- Allen Bradley;
- Bailey;
- Fisher Controls;
- Foxboro;
- Honeywell;
- Moore Products;

- Alfa Laval Automation;
- Taylor;
- Toshiba;
- Turnbull;
- Yokogawa.

2.2.2 Problemas numéricos nos controladores e quantização do ponto de vista de controle

Os efeitos da quantização de dados começam pelo problema da finitude das representações das operações aritméticas (matemáticas) e dos sinais usados. O termo finitude é impregado no sentido de “o limite” da representação. Um exemplo deste caso é a constante pi (π) que não possui fim, mas que sempre que usada é necessário indicar o seu limite. As fontes de erros encontradas por esses elementos são (SANTINA, 1996):

- Coeficiente de quantização;
- Quantização do conversor A/D;
- Operações aritméticas.

O erro pode ser indicado pela equação 6.

$$\text{erro} = \mathrm{Q}[\mathrm{x}] - \mathrm{x} \tag{6}$$

Onde o Q[x] é o valor quantizado de “x” e o erro da quantização depende do tipo de aritmética usada bem como o tipo de técnica de quantização usada. A quantização pode ocorrer por truncamento ou arredondamento. Na figura 4a é apresentado o erro de quantização pelo complemento de 1, pelo complemento de 2 na figura 4b e o método de arredondamento é mostrado na figura 4c.

A figura 4 mostra o comportamento do erro para cada um desses casos acima. No eixo horizontal está o valor real e no eixo vertical o número quantizado.

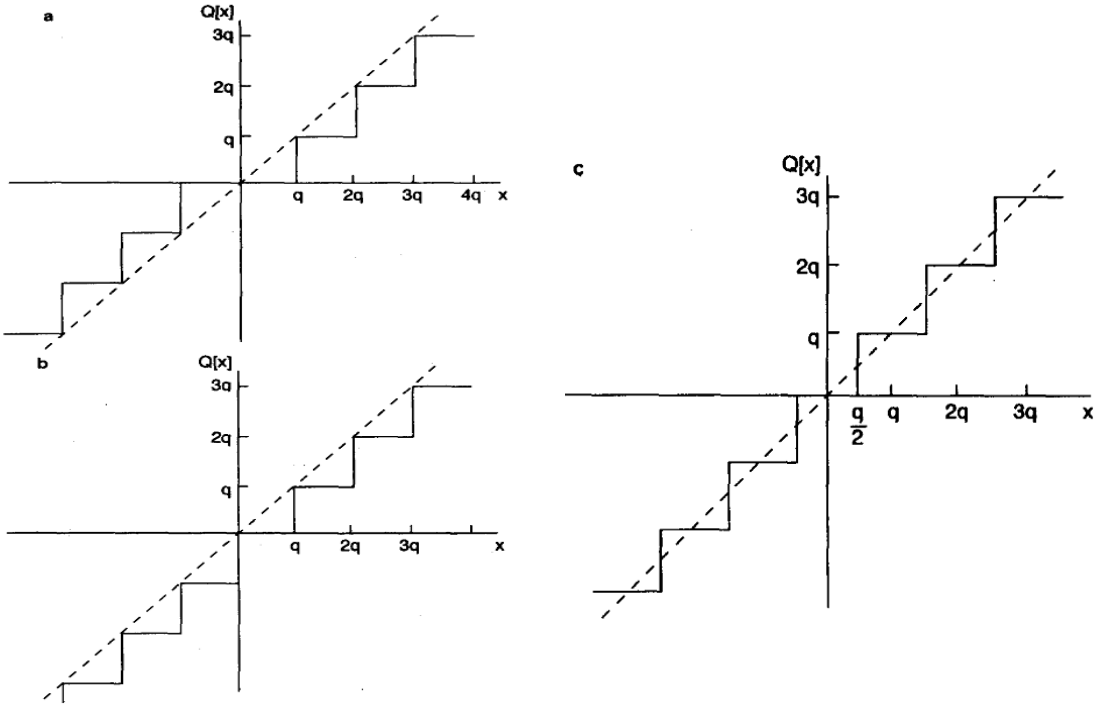


Figura 4 – Apresentação gráfica dos erros de truncamento e arredondamento. A) truncamento por complemento de 1, b) truncamento por complemento de 2 e, c) arredondamento.
Fonte: Santina *et al* (1996, p. 303).

Santina *et al*. (1996) exemplificam que o corte na precisão dos dados da planta+controlador geram pólos fora do círculo unitário, ou seja, a simplificação transforma o sistema discreto estável em um sistema instável do ponto de vista de controle (SANTINA, *et al*, 1996. p. 305).

As alterações não são lineares devido ao comportamento dos pólos. Por exemplo, uma alteração pequena em um sistema que possua pólos próximos afeta o comportamento final do sistema de forma mais agressiva quando comparado a um sistema com pólos distantes. Para evitar essa sensibilidade dos pólos uma possibilidade é decompor o sistema em cascata com funções de transferência de ordem mais baixa. Com isso esse problema possivelmente será minimizado. Caso o sistema possua pólos complexos é possível decompor em sistemas de segunda ordem (SANTINA et al, 1996).

Ainda com Santina *et al* (1996), caso seja necessário calcular a relação sinal ruído do conversor analógico para digital (A/D) é possível fazê-lo com base na equação 7.

$$\frac{\text{Sinal}}{\text{ruído}}(\text{db}) = 10\log_{10}\frac{\text{variância(sinal perfeito)}}{\text{variância(erro)}} \qquad (7)$$

$$= 10\log_{10} \frac{\text{variância}(x[k])}{\text{variância}(e)}$$

Entretanto, para alguns sistemas que possuem uma precisão maior ou igual a 12 bits, (para sinais inteiros) o erro de quantização do conversor A/D não é um problema sério, pois a precisão é significativa. Observe que caso essa precisão em bits seja aumentada o "ruído" gerado torna-se cada vez menor.

Por fim Santina *et al* (1996) comentam três itens importantes na área de controle:

- Quando controladores com limite de representação são implementados pode ocorrer o aparecimento de oscilações sustentadas ou ciclos limitados na saída do controlador. Isto ocorre quando não existe sinal de entrada aplicado, ocorrendo basicamente por dois motivos: o primeiro é pelo efeito do limite de banda passante (*deadband*) e o segundo é a extrapolação da representação, chamado em inglês de *overflow*.
- De forma direta e simplificada o sistema com baixa representação pode (possivelmente) acarretar em erro em regime permanente. Tal afirmação também é comentada por Astrom e Hugglund (1995).
- Quando se usa ponto fixo o erro de quantização ocorre com as operações de multiplicação e não com operações de adição. É importante observar essa operação matemática com mais cuidado.

No próximo subcapítulo da dissertação será comentado sobre a computação reconfigurável que é a plataforma de estudo proposta.

2.3 COMPUTAÇÃO RECONFIGURÁVEL

A computação reconfigurável começou com os PLDs (*programmable logic devices*) que foram criados em meados de 1970 com a proposta da construção de circuitos combinacionais lógicos que fossem programáveis. Os PLDs se diferenciam dos microprocessadores por não possuírem *hardware* fixo, ou seja, ele é configurável.

Segundo Pedroni (2010) em 1980 foram agrupados a esse dispositivo flip-flops, portas lógicas e multiplexadores. A tecnologia foi melhorando até o ponto atual

onde a divisão é feita entre unidades lógicas que representam uma macrocélula que contém as operações disponíveis. Essas unidades lógicas são agrupadas para desenvolver a função desejada. Com o avanço da tecnologia as PLDs foram chamadas de CLPDs, que são PLDs “Complexas”. Hoje (2012) eles possuem ferramentas de desenvolvimento sofisticadas com padronização pelo IEEE (*Institute of Electrical and Electronics Engineers*).

O termo FPGA (*Field Programmable Gate Array*) foi introduzido pela empresa Xilix que alterou as antigas CPLDs em tecnologia, arquitetura, tamanho, custo e desempenho, caracterizando um salto tecnológico.

Durante a pesquisa dessa dissertação foram encontradas duas grandes empresas desenvolvedoras e vendedoras de produtos com FPGAs: Xilix e Altera. Entretanto existem outras empresas menores como Atmel, Lattice e QuickLogic.

Maxfield (2004) descreve a arquitetura da FPGA como sendo basicamente um conjunto de blocos lógicos com conexões programáveis entre os blocos. Essa representação simplificada é apresentada na figura 5.

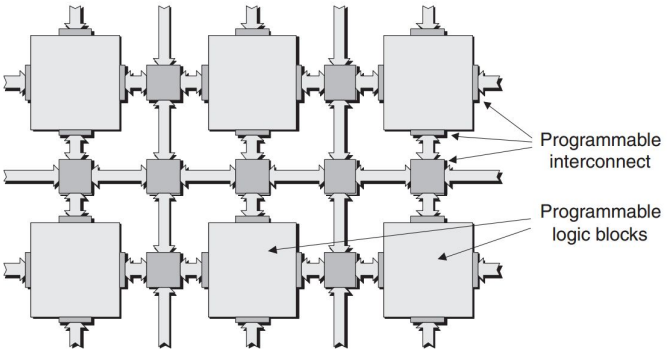


Figura 5 - Apresentação dos blocos lógicos das FPGAs
Fonte: MAXFIELD (2004, p. 52).

Dependendo do fabricante eles chamam suas unidades lógicas de CLBs (C*onfigurable Logic Blocks*) ou LABs (*Logic Array Blocks*) nessa dissertação será usado o termo genérico unidade lógica ou elemento lógico (EL) para não indicar preferência por algum fabricante em particular.

Em aplicações modernas como processamento de áudio e vídeo as FPGAs podem contar com blocos DSP (*digital signal processing*) que ajudam no tratamento desses dados mediante uso de filtros digitais que são concebidos com multiplicadores, acumuladores (somadores) e registradores que variam sua velocidade de processamento com base na tecnologia usada na construção das FPGAs bem como o *clock* usado. Segundo Pedroni (2010) o *clock* distribuído e

manipulado nas FPGAs é um elemento crítico, pois podem existir tempos de clock diferentes ao longo da placa, problema conhecido como *clock skew*.

A grande pergunta é por que usar FPGAs em projetos de controle? A resposta é: eles possuem alta velocidade, custo decrescente e vários softwares de desenvolvimento, incluindo integração com ferramentas de controle clássicas como Matlab® e seu pacote de simulação Simulink®, que são muito importantes na área de controle e identificação de sistemas dinâmicos e servomecanismos. Além disso, existe a possibilidade de operação de vários blocos de forma paralela com altas velocidades de operação.

2.3.1 Linguagens de descrição de hardware

Algumas linguagens de descrição de hardware para FPGAs são o VHDL e o Verilog. Nesse trabalho será apresentado apenas a VHDL por ser a linguagem utilizada pelo grupo de pesquisa ao qual o autor está vinculado. VHDL significa VHSIC *Hardware Description Language* que é uma linguagem padronizada pelo IEEE para a descrição de *hardware* (IEEE, 2000).

Existem algumas empresas que produzem *softwares* para a síntese da VHDL, são elas:

- Altera: Quartus II (síntese e simulação).
- Xilinx: ISE (síntese e simulação).
- Mentor Graphics: Precision (síntese) e Modelsim (simulação, esse último em especial possui integração com o simulink em sua versão completa).
- Synopsys: Design Compiler Ultra, VCS e Synplify pro/Premiere.
- Cadence: NC-Sim (simulação).

De forma simplificada a linguagem indica como o *hardware* (unidades lógicas) devem se comportar e o sintetizador projeta o arranjo físico das unidades lógicas.

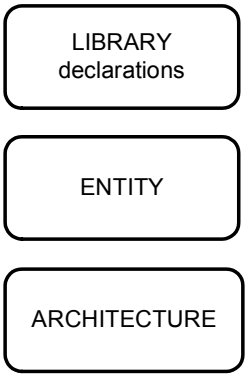
A estrutura do código em VHDL passa por três etapas, são elas (PEDRONI, 2010 e 2004):

- Declaração das bibliotecas: nesse ponto são feitas as chamadas de pacotes e bibliotecas utilizadas na compilação. Existem bibliotecas padronizadas e

existe a possibilidade da utilização de pacotes específicos criados exclusivamente para cada novo projeto;

- Descrição das entidades: basicamente é a descrição das entradas e saídas utilizadas no projeto sendo a quantidade e tipo de dados usados naquela porta ou conjunto de portas. Existe a possibilidade da criação de variáveis genéricas (globais) nesse espaço;
- Arquitetura do programa: nesse último tópico está o código propriamente dito onde a estrutura pode executar os cálculos de forma serial ou paralela (concorrente).

A figura 6 apresenta um diagrama em blocos e ao lado um exemplo prático da programação em VHDL.




```
1  ---------------------------------------
2  ENTITY example IS
3     PORT ( a, b, clk: IN BIT;
4            q: OUT BIT);
5  END example;
6  ---------------------------------------
7  ARCHITECTURE example OF example IS
8     SIGNAL temp : BIT;
9  BEGIN
10    temp <= a NAND b;
11    PROCESS (clk)
12    BEGIN
13       IF (clk'EVENT AND clk='1') THEN q<=temp;
14       END IF;
15    END PROCESS;
16 END example;
17 ---------------------------------------
```

Figura 6 - Blocos básicos de um programa e um exemplo em VHDL
Fonte: Pedroni (2010, p. 12).

De forma simplificada os tipos de dados usuais em FPGA estão na tabela 1. Onde a área escura (em cinza) apresenta os tipos de dados do VHDL 2008, o qual ainda está em processo de regulamentação e é um dos elementos essenciais dessa dissertação.

Tabela 1 – Resumos dos pacotes e dados usados em VHDL

Dimensão	Tipo predefinido	Biblioteca ou pacote de origem	Valores sintetizáveis sem restrições
Escalar	booelan	Std/standar	True, false
	bit	Sdt/standar	‘0’, ‘1’
	Std_(u)logic	IEEE/std_logic_1164	Estrada: ‘0’, ‘1’, ‘L’, ‘H’ Saída: : ‘0’, ‘1’, ‘L’, ‘H’, ‘X’, ‘W’, ‘Z’
1D	Bit_vector	Std/standar	Idem bit
	Std_(u)logic_vector	IEEE/std_logic_1164	Idem std_(u)logic
	Unsigned, signed	IEEE/numeric_std ou IEEE/std_logc_arith	Idem std_(u)logic

	integer	Sdt/standar	-(231-1) a (231-1) padrão
	natural	Sdt/standar	0 a (231-1) padrão
	positive	Sdt/standar	1 a (231-1) padrão
	character	Sdt/standar	Código ASCII estendido
1D x 1D	string	Sdt/standar	Conjunto de caracteres
1D	Boolean_vector	Sdt/standar	Idem boolean
	Ufixed, sfixed (propostos)	IEEE/fixed_pkd + pacotes associados	Valores numéricos de ponto fixo
	Float (propostos)	IEEE/float_pkd + pacotes associados	Valores numéricos de ponto flutuante (IEEE 754)
1D x 1D	Integer_vector	Sdt/standar	Idem integer

Fonte: adaptado de Pedroni (2012, p. 440)

Em VHDL é possível trabalhar com as operações lógicas, operações aritméticas, operações de comparação, operações de deslocamento (*shift*), operações de concatenação e operações de atribuição. No presente trabalho, que envolve a implementação de controladores PID usando o VHDL 2008, o foco será nos operadores matemáticos. Os tipos de operadores e operações associadas são apresentados na tabela 2.

Tabela 2 – Operadores básicos e tipos predefinidos em VHDL

Tipos de operadores	Operadores predefinidos	Tipos de dados sintetizáveis
Lógicos	Not, and, nand, or, nor, xor, xnor	Boolean, bit, bit_vector, std_(u)logic, std_(u)logic_vector, (un)signed
Aritméticos	+, -, *, /, **, abs, REM, mod	Integer, natural, positive, (un)signed, std_(u)logic_vector
Comparação	=, /=, >, <, >=, <=	Boolean, bit, bit_vector, integer, natural, positive, (un)signed, character, string, std_(u)logic_vector
Shift	sll, srl, sla, sra, rol, ror	Bit_vector, std_(u)logic_vector, (un)signed
Concatenação	& (também “,” e others)	Bit_vector, std_(u)logic_vector, string, (un)signed

Fonte: adaptado de Pedroni, 2010, p.448

No momento da programação em VHDL uma pergunta fundamental deve ser respondida: o código será paralelo (concorrente) ou em série (sequencial)? Mediante esta resposta é que são selecionadas as instruções passíveis de uso. Para cada caso existem instruções específicas. Como exemplo, no código concorrente, são usados os comandos WHEN, SELECT, e GENERATE, enquanto que no código sequencial, são usados os comandos IF, CASE, LOOP e WAIT.

Dentro dos padrões do VHDL, existem tópicos passando por análise. Entre eles, o ponto fixo e ponto flutuante no VHDL 2008. Neste trabalho, o ponto fixo será fundamental para o tratamento numérico na área de controle de sistemas dinâmicos e servomecanismos.

2.4 CÁLCULOS NUMÉRICOS EM FPGA COM VHDL 2008

O VHDL 2008 está em processo de padronização, entretanto ele já é discutido em alguns livros, artigos e páginas da internet. O foco principal do VHDL 2008 é melhorar algumas rotinas antigas e estabelecer padrões para operações aritméticas de ponto fixo e de ponto flutuante conforme a norma IEEE 754 (BISHOP, 2012).

O VHDL 2008 esta sendo especificado como IEEE 1076-2008. Mais informações podem ser encontradas no *Standard VHDL Language Reference manual*. Entretanto, nesse trabalho serão usados conceitos apresentados por Pedroni (2009-2010) e Bishop (2012).

Segundo David Bishop (2012) o ponto fixo é a representação intermediária entre os números inteiros e os números com ponto flutuante. A representação em ponto fixo é uma boa opção, desde que os números não variem muito em magnitude. Observando-se esta ressalva, a utilização da representação numérica em ponto fixo em FPGAs é perfeitamente possível. Por outro lado, a representação em ponto flutuante sempre pode ser utilizada, levando-se em conta a maior demanda por recursos da FPGA em relação à representação em ponto fixo. O pacote desenvolvido por Bishop deve ser usado com o VHDL-93 (1993).

Para o funcionamento é necessário o uso dos seguintes pacotes disponíveis em www.vhdl.org/fphsl/vhdl.html:

- Fixed_float_types.vhdl;
- Fixed_generic_pkg.vhdl;
- Fixed_generic_pkg-body.vhdl;
- Fixed_pkg.vhdl.

Esses pacotes foram desenvolvidos para uso no VHDL 2008, e provavelmente estarão na nova versão das bibliotecas do IEEE. Além dos pacotes apresentados acima é necessária a utilização de mais dois pacotes da versão VHDL 93 para compatibilidade, são eles:

- Fixed_float_types_c.vhdl: trata dos tipos de números com ponto fixo usados no pacote;
- Fixed_pkg_c.vhdl: versão de compatibilidade para o VHDL 93.

Tendo em mãos esses pacotes é necessário definir o seu domínio, ou seja, se serão usados números com ou sem sinal, “sfixed” ou “ufixed”, respectivamente.

Um exemplo descrito por Bishop (2012) é:

```
use ieee.fixed_float_types.all; -- ieee_proposed for VHDL-93 version
use ieee.fixed_pkg.all; -- ieee_proposed for compatibility version
....
signal a, b : sfixed (7 downto  -6);
signal c: sfixed (8 downto  -6);
begin
....
c <= a + b;
```

Devido à distinção entre representação com sinal e sem sinal, foi criada a função de conversão entre representações. As conversões possíveis são entre representações de números inteiros (*integer*), de números reais (*real*), de números com sinal (*signed*) e de números sem sinal (*unsigned*). Bishop (2012) descreve alguns exemplos de uso:

```
a <= to_sfixed (-3.125, 6, -6); -- transforma a constante "-3.125" em um número com sinal, com 6 bits antes e 6 bits depois do ponto.
b <= to_sfixed (inp1, b); -- transforma “inp1” em um número com as mesmas características de “b”.
y <= to_ufixed (6.5, 4, -5); -- onde “4” representa o número de bits antes do ponto e  “-5” representa o número de bits depois do ponto.
y <= to_ufixed (6.5, y'high, y'low); -- y'high e y'low representam variáveis contendo o número de bits antes e depois do ponto, respectivamente.
```

Segundo Bishop (2012) também existe a possibilidade de operações em apenas uma área da região numérica sem passar pelo zero, ou seja, operação ou com números positivos ou com números negativos. Por exemplo:

```
signal z : ufixed (-2 downto -3);
z <= "11"; -- 0.011 = 0.375
signal x : sfixed (3 downto 1);
x <= "111"; -- 1110.0 = -2
```

No caso de operações entre números representados por ponto fixo, o tamanho final dos campos, antes e depois do ponto, dependerá do tipo de operação

e do tipo de representação. Por exemplo, a operação de multiplicação de números em ponto fixo sem sinal gerará um número também representado por ponto fixo sem sinal, mas com número de bits antes e depois do ponto, dados pela somas de cada elemento antes do ponto e a soma de cada elemento depois do ponto, acrescido de um (1). Deste modo:

```
signal x : ufixed (7 downto -3);
signal y : ufixed (2 downto -9); -- Se houver a multiplicação entre esses números (x*y) o tamanho do número final será, ufixed (10 downto -12):
x * y : ufixed (7+2+1 downto -3+(-9));
```

Já no caso da representação com sinal o cálculo é diferente, por exemplo;

```
signal x : sfixed (-1 downto -3);
signal y : sfixed (3 downto 1); --Se houver a divisão entre os números x e y o tamanho final será sfixed (-1 downto -6).
x/y : sfixed (-1-1+1 downto -3-3)
```

A tabela 3 resume o cálculo dos campos para diferentes operações e representações com e sem sinal.

Tabela 3 – Forma de operação e conversão entre números com ponto fixo.

Operation	Result Range
A + B	Max(A'left, B'left)+1 downto Min(A'right, B'right)
A - B	Max(A'left, B'left)+1 downto Min(A'right, B'right)
A * B	A'left + B'left+1 downto A'right + B'right
A rem B	Min(A'left, B'left) downto Min(A'right, B'right)
Signed /	A'left - B'right+1 downto A'right - B'left
Signed A mod B	Min(A'left, B'left) downto Min(A'right, B'right)
Signed Reciprocal(A)	-A'right downto -A'left-1
Abs(A)	A'left +1 downto A'right
- A	A'left +1 downto A'right
Unsigned /	A'left - B'right downto A'right - B'left -1
Unsigned A mod B	B'left downto Min(A'right, B'right)
Unsigned Reciprocal(A)	-A'right +1 downto -A'left

Fonte: Bishop (2012). Disponível em: http://www.vhdl.org/fphdl/Fixed_ug.pdf

Alguns comandos são definidos para facilitar as operações, evitando o cálculo repetitivo de determinação do tamanho dos campos toda vez que as mesmas sejam realizadas. Os comandos são: resize, ufixed_high, ufixed_low, sfixed_high, sfixed_low. Como exemplo, existem quatro formas de uso (código em VHDL):

```
variable a : sfixed (5 downto  -3);
```

```
variable b : sfixed (7 downto  -9);
variable adivb : sfixed (sfixed_high (5,  -3, '/', 7, -9) downto sfixed_low (5, -3, '/', 7, -9)); -- determina o tamanho resultante dos campos para operação divisão. Trata-se da primeira forma.
Begin
adivb <= a / b; -- divisão simples (a operação);
Signal adivb : sfixed (sfixed_high (a'high, a'low, '/', b'high, b'low) downto sfixed_low (a'high, a'low, '/', b'high, b'low)); -- trata-se da segunda forma.
Signal adivb : sfixed(sfixed_high (a,'/',b) downto sfixed_low (a,'/',b)); --trata-se da terceira forma.
x <= resize (x / y, x'high, x'low); -- O comando resize pode ser usado para especificar o tamanho, entretanto ele será arredondado e saturado.
```

Das quatro formas apresentadas por (BISHOP, 2012), optou-se nessa dissertação pelo uso do comando *resize*, por ser a representação mais simples.

A forma como os números são tratados pelo compilador também pode ser alterada. O VHDL 2008 permite a alteração da forma de arredondamento, de truncamento, de controle de overflow (estouro de memória ou saturação) e de indicação de alerta (*warning*). Todas essas características podem ser controladas dentro do pacote fixed_pkg, permitindo otimizações para cada projeto. Entretanto o autor (BISHOP, 2012) observa a relação entre as alterações escolhidas e a quantidade de unidades lógicas utilizadas. Por exemplo, o uso do arredondamento demanda mais unidades lógicas do que o uso do truncamento (Bishop 2012, p. 6).

2.4.1 Ponto flutuantes no VHDL 2008

A vantagem do ponto flutuante sobre o ponto fixo está na variabilidade das amplitudes. Isto permite, durante uma série de operações, que a representação em ponto flutuante mantenha a precisão enquanto que a representação em ponto fixo perca a precisão. Os pacotes de implementação de ponto flutuante são diferenciados, assim como os pacotes de ponto fixo.

As conversões entre tipos de dados também existem para as operações com ponto flutuante. Como as mesmas não serão utilizadas nesta dissertação, não serão descritas. Entretanto, para mais informações recomenda-se a leitura do artigo de Bishop (2012).

Segundo o FAQ (*Frequently Asked Questions*) do site (http://www.eda.org/fphdl/fpfaq.html) do EDA (EDA Industry Working Groups, 2012) o processo de redimensionamento de variáveis constantemente é fonte de problemas para os programadores. Por este motivo o pacote encontrado em (http://www.vhdl.org/fphdl/fixed_noresize.vhdl) que opera com a base numérica padrão (standard) do VHDL 93 do tipo sinalizado e não sinalizado (*signed and unsigned*) é indicado para operação em ponto flutuante.

Como a proposta da dissertação é trabalhar com ponto fixo, buscando uma solução de compromisso entre a demanda por recursos da FPGA e precisão dos controladores, as questões que envolvem o ponto flutuante não serão comentadas a fundo.

O tópico seguinte discorre sobre a implementação de controladores em FPGA, suas limitações, suas vantagens e procedimentos de programação.

2.5 RECOMENDAÇÕES SOBRE A IMPLEMENTAÇÃO DE CONTROLADORES EM FPGA

Segundo Burden e Faires (2001) o procedimento para obtenção de um resultado usado em uma calculadora ou em computador é diferente do procedimento da aritmética, da álgebra ou do cálculo convencional. Para um ser humano a relação $\left(\sqrt{3}\right)^2 = 3$ é verdadeira. Já para uma máquina não é tão simples, fazendo-se necessária a pesquisa sobre a finitude dos dígitos e representação dos números.

Burdens e Faires (2001) ainda comentam sobre os algoritmos e suas formas de análise de erro ao longo de várias interações, situação idêntica à realizada pelos controladores que atuam sobre plantas reais a cada nova amostragem. Dessa forma, é necessário verificar se o processo de arredondamento de variável, por exemplo, não incrementa os erros de cálculo com o passar do tempo.

Todo o processo de implementação começa com o estudo dos erros possíveis dentro de sistemas limitados por operações ou precisão de representação, sistemas esses como as FPGAs. Segundo Cunha (2000. p. 19):

> *"Os métodos numéricos são utilizados para calcular uma aproximação para a solução desejada. Na escolha de uma nova solução numérica, abre-se um novo leque de*

procedimentos alternativos, e a cada um deles está associada uma nova fonte de erros."

Segundo a definição do autor (Cunha, 2000), os erros numéricos podem ser classificados como:

- Erro inicial: são as incertezas introduzidas no equacionamento do problema, na medição dos parâmetros e nas condições iniciais.
- Erro de truncamento: é o erro encontrado quando se aplica um truncamento (corte numérico) a um processo infinito. Dois exemplos desse erro são: a utilização de séries no cálculo de funções e o uso de equações a diferenças finitas para aproximar as equações diferenciais contínuas.

Segundo Cunha (2000) e Chapra (2001) existem basicamente dois tipos de medida de erros: o erro absoluto e o erro relativo. O erro absoluto é representado pelo valor real subtraído do valor representado, enquanto que o erro relativo é o erro absoluto dividido pelo valor exato. De forma a exemplificar numericamente esses erros, são apresentadas as equações 8 e 9, onde α e β são, respectivamente, o valor exato e o valor representado para um número qualquer.

$$erro\ absoluto = |\beta - \alpha| \tag{8}$$

$$erro\ relativo = \left|\frac{\beta - \alpha}{\alpha}\right| \tag{9}$$

Para representar o erro relativo de forma percentual basta multiplicar o valor obtido por 100 (%).

Ainda segundo Cunha (2000), define-se $\lambda_i, i = 1{:}k$ como as raízes da equação $f(\lambda) = 0$, derivada da equação 10, se $|\lambda_i| \leq 1,\ i = 1{:}k$ e todas as raízes com valor absoluto forem igual a 1 (se existirem) são raízes simples, ou seja, se $|\lambda_i| = 1$ e então $f'(\lambda_i) \neq 0$, então é verdade que o método satisfaz a condição de raiz.

O elemento λ é o elemento do polinômio característico, conforme a equação 10.

$$f(\lambda) = a_k\lambda^k + a_{k-1}\lambda^{k-1} + \cdots + a_1\lambda + a_0 \tag{10}$$

Assim, a estabilidade de um método de discretização para equações diferenciais é caracterizado pelas seguintes definições:

i. Se um método satisfaz a condição da raiz e λ=1 é a única possibilidade para as raízes de módulo 1, então ele é chamado fortemente estável;
ii. Os métodos que satisfazem a condição da raiz e têm mais de uma raiz com módulo igual a 1 são chamados fracamente estáveis;
iii. Os métodos que não satisfazem a condição da raiz são chamados instáveis.

Uma definição mais simples para análise é indicada por Burden e Faires (2001): Definição: Supondo que $E_0>0$ seja o erro inicial e que E_n represente a magnitude do erro após n interações subsequentes. Se $E_n \approx C_n E_0$, onde C é uma constante independente de n, então o erro crescente é linear. Se $En \approx C^n E_0$, para algum C>1 então o crescimento do erro é exponencial. O erro crescente de forma linear é comum e difícil de evitar, mas caso os valores de C e E_0 sejam pequenos o valor final é aceitável. Já no caso do erro exponencial ele é inaceitável devido à imprecisão gerada. Desta forma o erro exponencial cria um algoritmo instável. A figura 7 apresenta de forma gráfica o comportamento desses dois tipos de erros.

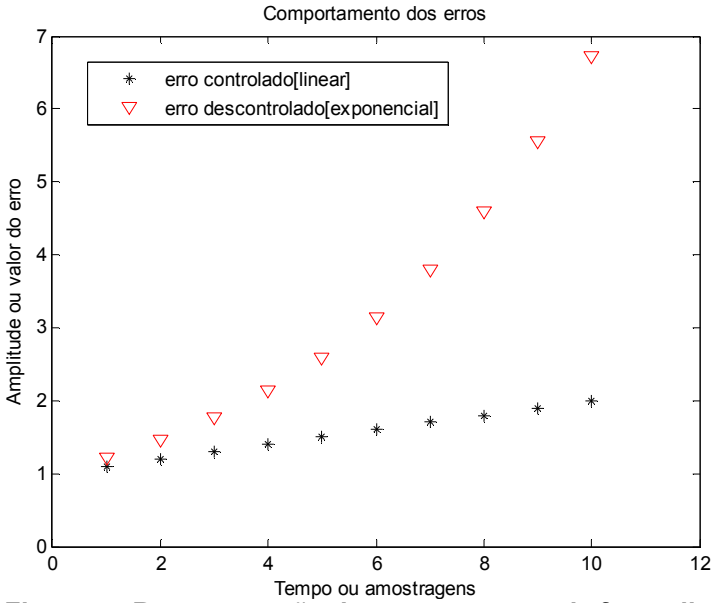


Figura 7 - Representação de erro crescente de forma linear e exponencial
Fonte: Adaptado de Burden e Faires (2001, p. 34).

Na busca pela solução desse problema, Chapra (2008) comenta algumas etapas de análise que devem ser levadas em consideração. Para o autor, na era atual, os problemas numéricos passam por três etapas para a sua solução (CHAPRA, 2008. p. 3):

- A solução é alcançada usando algum método analítico ou exato. Entretanto soluções analíticas estão relacionadas fortemente com sistemas lineares, o que torna sua aplicação limitada.
- Soluções gráficas que tentam caracterizar o sistema. Podem ser aplicados a sistemas complexos, mas geram imprecisão. Além disso, se limitam a problemas de três dimensões ou menos, que é a quantidade máxima representável de forma gráfica.
- Calculadoras e cálculos manuais. Os cálculos manuais são lentos e tediosos além de um simples erro nos cálculos iniciais comprometem todo o resto do trabalho.

Para Chapra (2001) os problemas numéricos da engenharia possuem na computação um divisor de águas, propondo a divisão entre as eras da pré-computação e da computação. Na era da pré-computação era necessário analisar as leis físicas que regem os sistemas e preparar um método adequado para ser usado na busca pela solução. Já na era da computação as leis são inseridas integralmente no computador, o qual realiza todos os cálculos necessários em um tempo muito menor em comparação ao tempo despendido por um ser humano fazendo os mesmos cálculos. A figura 8 mostra a relação entre os métodos de solução de problemas na engenharia com e sem computadores. O tamanho das caixas indica o nível de ênfase.

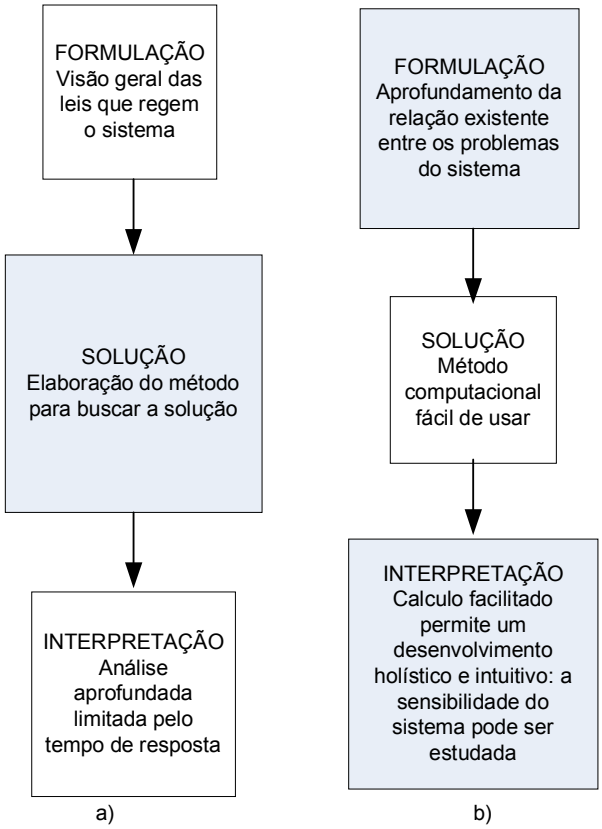


Figura 8 – Processo de solução de problemas numéricos, pré-computação (a), e na era da computação (b)
Fonte: adaptado de Chapra (2001, p. 4).

Os computadores são ferramentas de alta potência para a solução de problemas na engenharia, mas podem se tornar imprestáveis sem um conhecimento detalhado sobre como os sistemas funcionam (Chapra, 2001 p. 11). O autor ainda comenta que na vida profissional alguns erros podem custar caro ou serem catastróficos. Se estruturas ou dispositivo falharem vidas podem ser perdidas (Chapra, 2001 p. 50).

A grande pergunta a ser feita é: qual é a quantidade de erro presente no cálculo e quanto de erro é aceitável? Essa dissertação irá expor os erros encontrados pela diminuição da precisão de casas decimais. Tal apresentação é descrita nos próximos capítulos e/ou apêndices.

Dentro da análise sobre o erro é importante ressaltar a diferença entre a exatidão e a precisão. Para Chapra (2001) a exatidão representa o quão próximos do valor real estão os dados alcançados, enquanto a precisão refere-se à aproximação dos dados entre si, ou seja, distância do indivíduo para o conjunto. A figura 9 demonstra isso de forma gráfica.

A exatidão dos dados deve existir para que o problema possa ser suficientemente resolvido enquanto que a precisão é necessária para o projeto adequado.

A figura 9 apresenta de forma gráfica 4 possibilidades envolvendo a exatidão e a precisão. Na figura 9a existe baixa exatidão e baixa precisão. Na figura 9b existe baixa precisão e alta exatidão, com isso os dados coletados estão todos próximos do alvo, mas distante entre si. Na figura 9c existe baixa exatidão e alta precisão, com isso os dados estão próximos, mas longe do alvo. Na figura 9d existe alta exatidão e alta precisão, com isso os dados estão todos próximos do alvo e também próximos entre si.

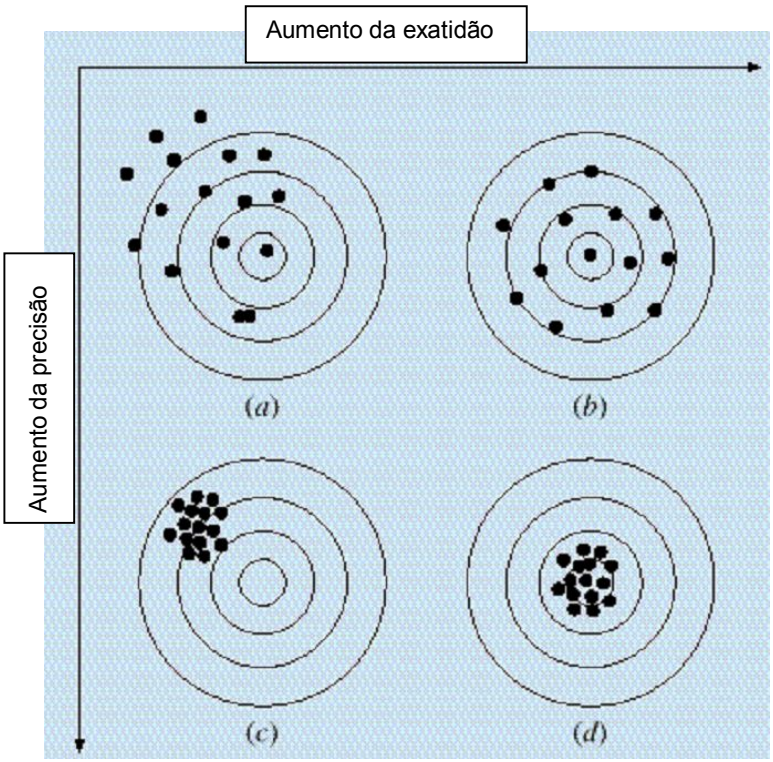


Figura 9 - Representação gráfica da precisão e da exatidão (*precision and accuracy*)
Fonte: Chapra (2001, p. 53).

Cavanagh (1985) comenta sobre o cuidado nas operações com ponto fixo, onde o produto possui duas vezes o tamanho dos operandos para manter a precisão (Cavanagh, 1985, p. 137). Esse ponto é muito importante, pois em casos com cálculos recursivos algum método de redimensionamento deve ser usado para não gerar espaços de memória crescentes.

Existe um método de multiplicação que usa as memórias do tipo RAM (*Random-Access Memory*), essa técnica é chamada de *table lookup multiplication*. Dentro dessa técnica existem duas possibilidades: na primeira existe o deslocamento para a direita de três bits dos números após a sua operação de soma.

Na segunda existe o deslocamento de quatro bits para a direita após a soma dos números. Segundo Cavanagh (1985), o último método é mais rápido, mas requer mais *hardware* para essa operação. Essas duas técnicas são variações do método padrão de adição-deslocamento (*add-shift*).

Já no caso da divisão também existe a operação de troca, mas agora é chamada de troca e subtração/adição (shift-subtract/add). A possibilidade de *overflow* continua (estouro do limite de memória). Também existe a implementação da técnica de Newton-Raphson que é um método iterativo onde a relação de divisão A/B é trocada por A*(1/B). Segundo Bishop (2012), esta técnica será implementada em VHDL 2008. Também existe a possibilidade de se usar divisores dedicados de alto desempenho.

De forma bem simples e direta, Cavanagh (1985) define que a aritmética usada em sistemas decimais é parecida com a usada em sistemas de ponto fixo. A principal diferença é que a aritmética de ponto fixo trata cada bit como sendo um dígito (CARVANAGH, 1985, p. 305).

Em Moreira e Farrel (2006) são discutidos os erros na transmissão de dados digitais que poderiam ser incorporados ao trabalho. Entretanto, como não foi constatado nenhum tipo de erro de transmissão, as técnicas de análise não serão comentadas. Em futuras aplicações, caso seja constatado que houve erro na transmissão, serão consideradas as técnicas indicadas nesse livro.

Depois dessa introdução sobre o cálculo numérico, tratar-se-á dos controladores discretizados. Os controladores podem ser chamados também de compensadores ou simplesmente de filtros.

2.5.1 Cuidados com o projeto de controladores

Segundo Phillips e Nagle (1995) existem alguns cuidados a serem tomados no projeto de controladores para que eles não entrem em uma condição de trabalho indesejado. Os problemas de controle estão relacionados, mas não limitados a:

- Rejeição a distúrbios;
- Erro zero em estado estacionário;
- Resposta transiente adequada;
- Sensibilidade à mudança nos parâmetros da planta.

A solução do problema de projetar um bom controlador envolve:

1. Encontrar os sensores para medição de variáveis do sistema e entregar os dados necessários;
2. Encontrar os atuadores para controlar a planta;
3. Equacionar o sistema envolvendo planta, atuadores e sensores (modelo matemático);
4. Projetar o controlador baseado nos critérios de controle necessários;
5. Avaliar o projeto por simuladores e, após esse processo, testar o controlador no sistema real;
6. Realizar todo o processo novamente até encontrar uma solução satisfatória.

Para que essas etapas tenham uma representatividade é necessário indicar a qualidade do projeto e isso passa por especificações de projeto. Dentro das especificações de controle do sistema existem os seguintes elementos (PHILLIPS e NAGLE, 1995, p. 282):

- Precisão de estado estacionário;
- Resposta transiente;
- Estabilidade relativa;
- Sensibilidade à mudança de parâmetros da planta;
- Rejeição a distúrbio;
- Esforço de controle: este, em especial, relata a quantidade de “energia” a ser disposta pelo atuador.

Phillips e Nagle (1995) comentam que pode existir uma conversão entre as formas de representar sistemas de controle e suas plantas. A figura 10 exemplifica essa conversão entre os dados. Nesta é indicado que uma função de transferência contínua pode ser transformada em uma função de transferência discreta, e esta última pode ser representada como uma equação a diferenças. O processo inverso também é possível.

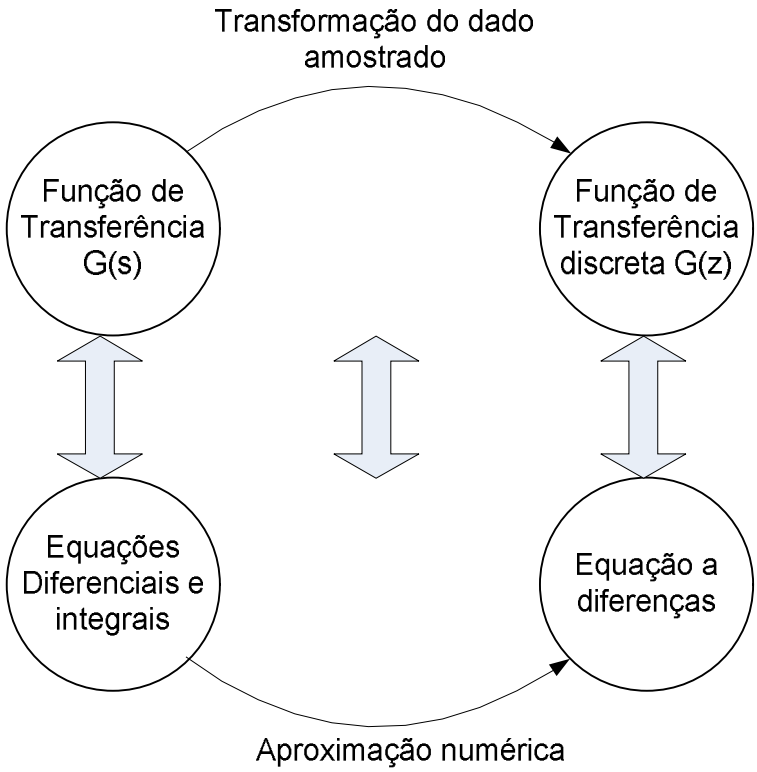


Figura 10 – Conversão de dados amostrados (A/D)
Fonte: adaptado de Phillips e Nagle, 1995, p. 431.

As técnicas de aproximação numérica para amostragem de dados são:

- *Backward difference*: é uma técnica simples onde se usa a diferença entre o termo atual y(t) e o termo anterior y(t-Ts), onde Ts é a diferença de tempo entre as amostras, conforme a equação 11.

$$\frac{d}{dt}y(t) \cong \frac{y(t) - y(t - Ts)}{Ts} \tag{11}$$

- *Forward difference*: é similar ao anterior, onde se usa a diferença entre o termo posterior y(t+Ts) e o termo atual y(t). A aproximação é mostrada na equação 12.

$$\frac{d}{dt}y(t) \cong \frac{y(t + Ts) - y(t)}{Ts} \tag{12}$$

- *Left-side rule*: nesse caso o componente da integração é estimado mediante um mapeamento de pontos no plano “s” em pontos no plano “z”. Desta forma os elementos de “s” (Laplace) são transformados em “z” segundo a equação 13.

$$\frac{1}{s} \cong \frac{Ts}{z - 1} \tag{13}$$

- *Right-side rule*: assim como o anterior a aproximação é do tipo retangular, entretanto agora ela está adiantada. A equação 14 mostra este mapeamento.

$$\frac{1}{s} \cong \frac{Ts}{1 - z^{-1}} \tag{14}$$

- *Trapezoidal rule*: essa técnica trabalha com a média entre o lado direito e esquerdo de um trapézio formado dentre os dados amostrados. Novamente, a conversão é feita sobre o elemento integrador do sistema. Essa conversão também é chamada de conversão bilinear. A equação 15 mostra a conversão.

$$\frac{1}{s} \cong \frac{Ts}{2} \cdot \frac{1 + z^{-1}}{1 - z^{-1}} \tag{15}$$

- *Simpson´s rule*: essa técnica é pouco usada. Ela consiste no aumento da ordem da aproximação. A equação 16 mostra a conversão.

$$\frac{1}{s} \cong \frac{Ts}{3} \cdot \frac{1 + 4z^{-1} + 4^{-2}}{1 - z^{-2}} \tag{16}$$

- *Impulse invariance*: essa técnica supõe que se deseja fazer uma conversão do sistema para uma entrada do tipo impulso. Ela é representada pela clássica conversão de Laplace que pode ser calculada para cada termo ou, de forma mais comum, podem ser usadas tabelas de conversão com as funções mais encontradas na área de controle discreto.
- *Impulse invariant integrator*: essa técnica é aplicada para conversão de sistemas digitais com alta velocidade, ou seja, para baixos valores entre os tempos de amostragens, caso isso seja real, ela é aplicável conforme a equação 17.

$$\frac{1}{s} \cong \frac{Ts}{1 - z^{-1}} \tag{17}$$

- *Step invariance*: essa técnica é aplicada em duas partes. Na primeira é necessário fazer a transformada “z” do sistema e após isso multiplexar o sistema pelo equivalente ($1\text{-}z^{-1}$). A técnica procura introduzir distorções provocadas pela conversão D/A presente na malha de controle. A equação 18 mostra essa conversão.

$$D(z) \cong (1 - z^{-1}) \cdot {}_{z}\left[\frac{G(s)}{s}\right] \tag{18}$$

Cada uma dessas técnicas de conversão acaba trazendo consequências no processo de discretização. Phillips e Naggle (1995) comentam sobre o mapeamento dessas conversões sobre o sistema. Para fins de exemplo, através de simulações em Matlab, é apresentado no apêndice A dessa dissertação os comportamentos de

alguns dos métodos de discretização e algumas observações sobre os mesmos no processo de discretização.

Segundo Phillips e Nagle (1995, p. 531) o processo de arredondamento causa uma precisão de 1.6 vezes maior que o processo de truncamento quando eles são analisados com o mesmo número de casas decimais. Do ponto de vista da implementação em VHDL 2008 isto representa uma maior demanda por unidades lógicas, principalmente devido aos bits de controle do arredondamento, mas com um ganho em precisão.

De outro modo, um filtro, eventualmente insensível a imprecisões nos seus coeficientes, quando acoplado em série ou paralelo tem uma grande chance de tornar instável um sistema de segunda ordem. Isto se deve ao fato das raízes do polinômio característico da planta serem mais sensíveis às mudanças dos parâmetros com o aumento da ordem do sistema (Phillips e Naggle, 1195, p. 541).

Algumas observações importantes de Phillips e Naggle (1995, p. 545):

- Quando os filtros digitais são usados para controle em malha fechada, os coeficientes do filtro têm um efeito decisivo sobre a execução do controle;
- Se as restrições de implementação de controladores forem conhecidas desde o início os controladores podem ser projetados para evitar os problemas de quantização;
- A quantização pode alterar as especificações do processo como margem de ganho e tempo de resposta.

O processo de quantização pode gerar erros tanto na sua entrada como na sua saída, ou seja, os conversores A/D e D/A podem se tornar fontes de problemas. No caso específico da saída de controle, a quantização pode ser um limite da amplitude da saída, de forma que o controle mantém erro em regime permanente.

Uma sequência interessante de projeto é comentada por Phillips e Naggle, (1995) onde os autores comentam sobre duas abordagens sobre a implementação de controladores em *hardware*. Na primeira, a implementação seria em paralelo e na segunda, em cascata (serial). A figura 11 resume os passos de projeto indicados pelos autores citados.

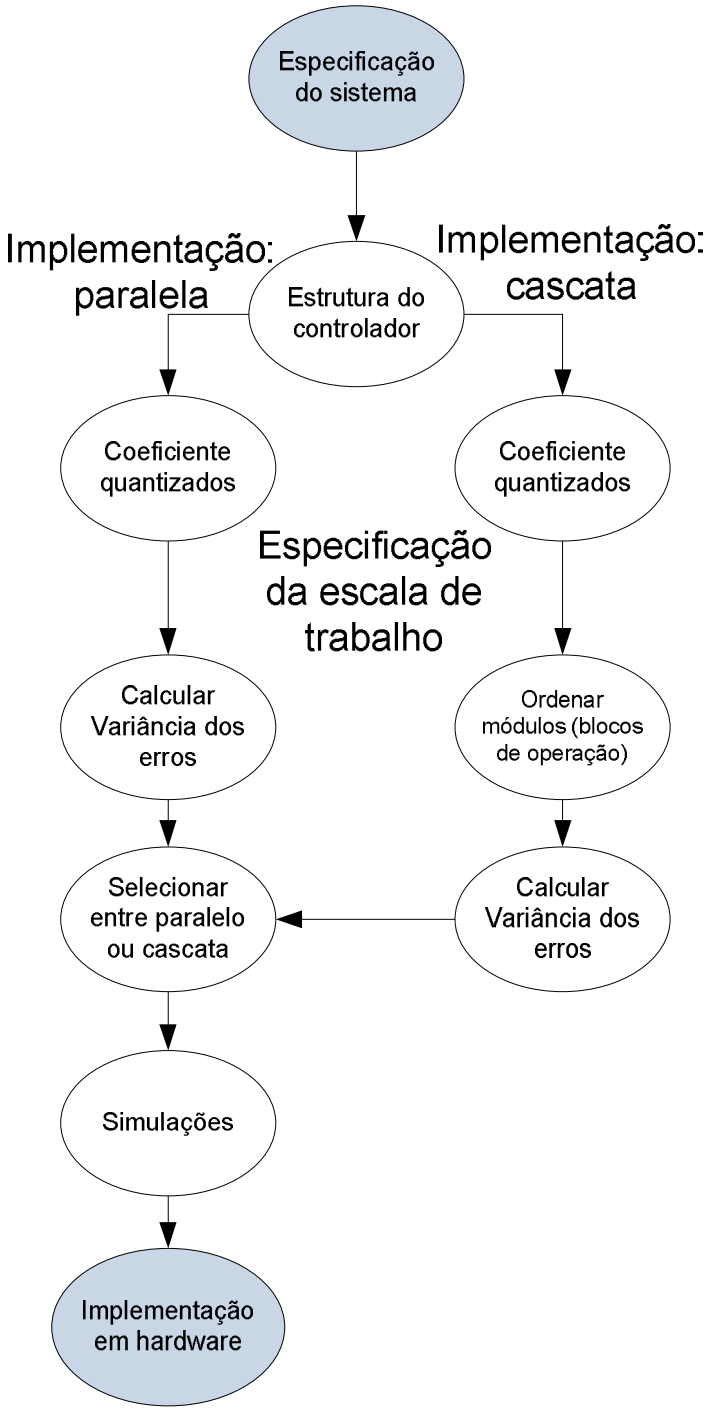


Figura 11 - Proposta de implementação de controladores em hardware de Phillips e Nagle
Fonte: adaptado de: Phillips e Nagle (1995, p. 592).

Para o caso específico do sistema em paralelo (lado esquerdo da figura 11), buscando uma otimização, o método fica na seguinte sequência (Phillips e Nagle, p. 592):

1. Expansão em frações parciais para otimização do controlador ou filtro;
2. Quantização dos coeficientes para determinar o limite numérico (*wordlenght*);
3. Verificação se a quantização dos coeficientes está de acordo com as especificações do sistema;

4. Achar a estrutura dos módulos;
5. Mudar a escala de cada módulo caso necessário. Com isso a implementação está completa;
6. Simular em malha aberta o filtro digital e testar com as entradas do tipo degrau, impulso e senoidal para ter a certeza que os limites dinâmicos de todas as variáveis são apropriados;
7. Inserir o filtro (controlador) digital dentro do sistema completo simulado para ter certeza que as especificações foram satisfeitas;

Essa metodologia de trabalho é interessante e será usada como referência nessa dissertação de forma a contribuir com o projeto de controladores em FPGA usando ponto fixo, mas apenas de forma serial mediante o uso de máquina de estados.

Como comentário importante, Phillips e Naggle (1995, p. 506) observam que existe a possibilidade de se gerar uma alocação de variáveis de forma mais eficiente. Isso é realizável com a implementação em paralelo dos coeficientes do filtro. Um exemplo ilustrativo seria um sistema de quarta ordem dividido em uma constante e mais duas equações de segunda ordem. Na época da publicação do livro (1995), a principal forma de processamento era serial. Nos dias atuais (2012), o processamento paralelo tornou-se uma opção viável, permitindo a realização de controladores antes apenas teóricos. Por outro lado, as FPGAs permitem, de forma paralela ou não, a execução de algoritmos simples de controle aplicados a sistemas com dinâmicas muito rápidas ou mesmo algoritmos complexos aplicados a sistemas com tempo de acomodação lento.

Em um de seus estudos de caso, Philips e Nagle (1995, p. 608) comentam que é comum o *hardware* possuir restrições predefinidas. Isto torna necessário o estudo e a adaptação das técnicas de controle usadas em casos ideais para sua aplicação em casos práticos.

Após esta breve revisão dos conceitos básicos, serão apresentados no próximo capítulo os trabalhos relacionados com o estado da arte de tópicos importantes para a dissertação.

3. TRABALHOS RELACIONADOS

Esse capítulo aborda o estado da arte encontrado na literatura sobre a aplicação de lógica reconfigurável ao controle de processos, particularmente controle discreto de processos, bem como a utilização do VHDL 2008 com ponto fixo.

De forma direta a busca recaiu sobre os seguintes itens:

- Controladores em FPGA;
- VHDL 2008;
- Ponto Fixo em VHDL;
- Desempenho e erro de controladores PID.

Monmasson e Cirtea (2007) apresentam um *survey* sobre a aplicação de FPGAs na área de controle de sistemas industriais. Os autores apresentam uma relação importante entre as restrições de arquitetura e desempenho de controle. Segundo os autores, o uso de controladores eficientes em FPGA ocorre devido a três fatores:

- Redução de custo: algumas metodologias diminuem o tempo de implementação e o desenvolvimento, permitindo inclusive a integração de grandes blocos com o desenvolvimento de sistemas em *chip* (SoC);
- Confidencialidade: é possível manter em segredo as técnicas de controle usadas;
- Aplicação crítica: sistemas embarcados com várias restrições como, por exemplo, consumo de energia, considerações térmicas, confiança e proteção contra eventos de distúrbios podem ser inplementados com FPGAs.
- Aperfeiçoamento do desempenho de controle: por exemplo, o tempo de execução pode ser drasticamente reduzido com projetos de arquitetura dedicadas em paralelo. Desta forma, permite-se um desempenho das FPGAs melhor que seus equivalentes analógicos, os quais possuem inconvenientes, como por exemplo, desvios de parâmetros e falta de flexibilidade.

Para Monmasson e Cirtea (2007) é possível, com o uso de HDL (*hardware description language*), aplicar um modelo holístico em plantas industriais. Um sistema hierárquico e modular para projetos em HDL é conhecido como projeto “top-down *approach*”. Essa metodologia consiste em quatro etapas para a criação de

códigos em um ambiente de desenvolvimento e simulação para HDL como VHDL ou Verilog, sendo eles:

- Nível de sistema: onde as especificações do sistema são dadas;
- Nível de ambiente: consiste na descrição do algoritmo na forma de circuito;
- Nível de transferência de registro (RTL): onde o circuito é “escrito” com base em seus componentes elementares;
- Nível físico: onde o circuito é fisicamente descrito levando em conta as características do *hardware* alvo (objetivo).

A figura 12 apresenta a arquitetura de projeto *top-down approach*.

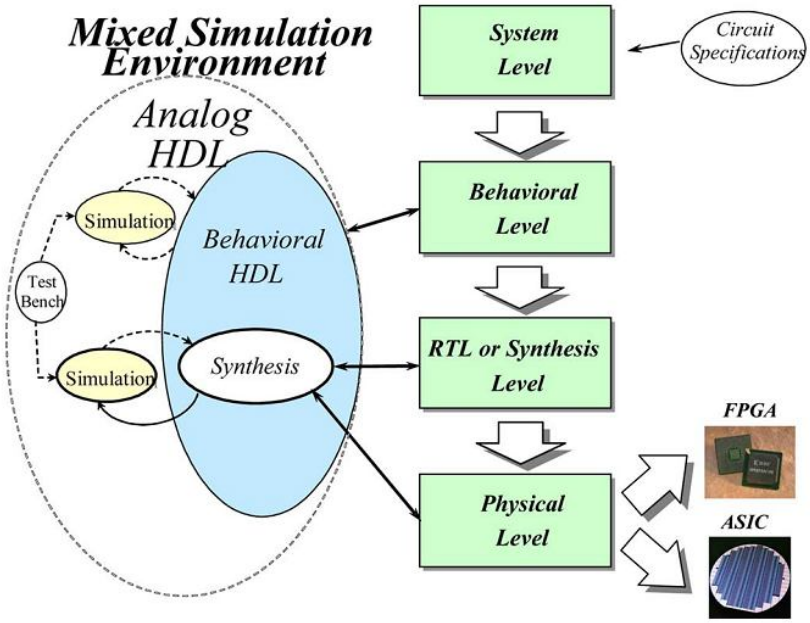


Figura 12 - Representação do projeto top-down approach
Fonte: Monmasson e Cirtea (2007, p. 3).

O artigo de Bustamente *et al* (2011) fala sobre a implementação de um controlador generalizado proporcional e integral (GPI) em comparação com um controlador proporcional integral derivativo (PID) para controle de um conversor DC-DC do tipo “Buck”. Os controladores foram implementados em FPGA. Segundo os autores, o controle GPI é interessante ao ser aplicado a sistemas conversores de energia. Isto se deve à rápida resposta dinâmica e robustez deste tipo de controlador. Explicam que, tipicamente, existem constantes desconhecidas ou distúrbios do tipo rampa no sistema. Além disso, esse controle tende a uma diminuição da quantidade de sensores necessários. Os tempos de processamento do GPI e PID foram de 39.2 e 20.54 us (microssegundos) respectivamente.

Como conclusão de Bustamente *et al* (2011), o controle GPI é melhor em desempenho e robustez. Entretanto, o controlador PID usou 8737 unidades lógicas e o GPI usou 19775, ambos com ponto flutuante.

Segundo Lai *et al* (2010), devido à velocidade, as FPGAs estão se tornando indispensáveis na aplicação de novos algoritmos de controle, como controle via lógica *fuzzy*, controle adaptativo, controle deslizante, entre outros. Um controlador PID multi canal em FPGA é implementado no artigo. Os parâmetros do PID foram limitados em 12 bits. O artigo não comenta sobre o erro ou variação existente entre os valores simulados e os reais, o que é ruim, pois apenas gráficos demonstrativos são apresentados. Além disso, os parâmetros PID selecionados foram estimados mediante a técnica de Ziegler e Nichols. Observando que essa técnica possui erros grandes quando comparados às técnicas atuais, consta-se um ponto mal explorado no artigo.

No artigo de Tang *et al* (2011) o ponto flutuante pode ser implementado de forma a converter dados de entrada com precisão. Segundo os autores, como justificativa para o uso de projeto em *hardware*, as FPGAs atuais apresentam grande volume de recursos, alta velocidade de operação e pequeno consumo de energia. Dessa forma, são largamente utilizadas no processamento digital de sinais que exigem alta velocidade de processamento.

Tang *et al* (2011) apresentam uma sequência, segundo a figura 13, para converter ASCII para ponto flutuante de precisão simples, projeto interessante caso o método de trabalho desconsidere o uso no VHDL 2008.

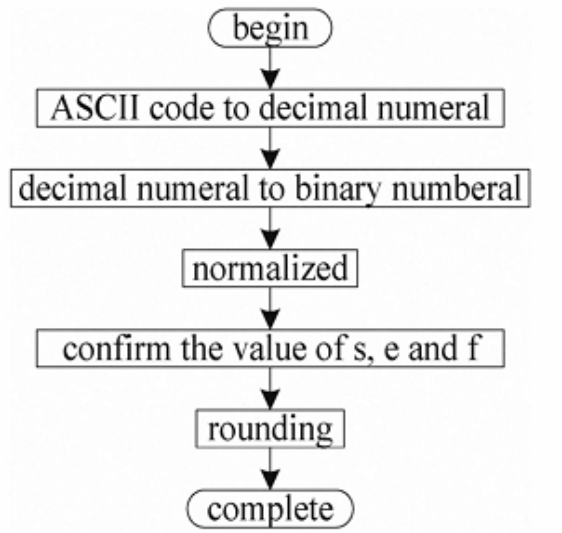


Figura 13 - Proposta de implementação ponto flutuante de Tang *et al* (2011)
Fonte: Tang *et al* (2011, p. 7).

O sistema de Tang et al (2011) foi simulado com o software ModelSIM da empresa Mentor Graphics.

No artigo de Thulasiraman et al (2010) o controle de motores de passo via rede wireless é implementado em FPGA. O projeto envolve a criação e um sistema de controle e supervisão de um motor de passo para distâncias de até 70 metros, mediante comunicação Wireless. O principal elemento controlado é a posição do motor de passo. A interface gráfica de usuário (GUI - *graphical user interface*) foi criada via LabVIEW da empresa National Instruments, mas as simulações e testes foram realizado via ModelSIM.

Khajehoddin *et al* (2010) implementaram um controlador ressonante em FPGA usando ponto fixo para seus cálculos. Existe um comentário muito interessante dos autores sobre o nível de precisão dos cálculos com base na velocidade da amostragem dos dados. Segundo eles, a precisão varia com a velocidade de amostragem e para ter uma boa precisão deve existir uma relação de 1 bit para cada 5KHz de amostragem. Segundo os autores, o método de discretização de Tustin é mais adequado como método de discretização. Outro dado interessante de Khajehoddin *et al* (2010) é que o erro do controlador aumenta quando o mesmo executa saltos entre frequências de amostragem ou de processamento. Recomendam que o controlador clássico mantenha sua taxa de amostragem fixa. No trabalho eles também indicam quantas unidades lógicas cada controlador utilizou. Isto é uma análise importante quando for necessário estimar a melhor relação custo benefício (custo VS. benefício) entre desempenho e demanda de recursos da FPGA.

O trabalho Zhang e Li (2008) é bem interessante na área de FPGAs pois trata do controle de motores de indução com redes neurais em *hardware*. Segundo os autores a FPGA e sua linguagem VHDL foram selecionados pela flexibilidade de programação, desempenho de velocidade, paralelismo e independência de fabricantes. Isto permite que o projeto seja rapidamente alterado para componentes de fabricantes diferentes ou mesmo outros componentes de um mesmo fabricante. Os autores citam que uma desvantagem das FPGAs é que usam muitas unidades lógicas no caso de operações aritméticas complexas, as quais podem ser realizadas com menor custo por DSPs, por exemplo.

Um comentário na revisão bibliográfica de Zhang e Li (2008) é que o erro nas operações matemáticas diminui com o aumento do clock, ou seja, com a diminuição entre o tempo de cada operação.

O artigo de Ide e Yokoyama (2004) relata a confecção de um controlador DeadBeat para um inversor PWM de três fases. Segundo os autores, o DSP possui limitações de velocidade que a FPGA não possui, uma vez que esta última pode executar funções em paralelo. Já o processamento serial do DSP pode acarretar atrasos de processamento prejudiciais ao funcionamento do controlador. Os autores comentam que usaram 500K gates para construção do controlador, permitindo uma taxa de amostragem de 1.15us. Entretanto não foi apresentada qualquer análise comparativa de erro entre o controlador em FPGA e outra opção em *software*.

Outros trabalhos justificam a utilização do *software* Matlab como ferramenta de pesquisa e desenvolvimento. Segundo Faldar *et al* (2001) o Matlab é mais popular em processamento de sinais e imagem por ser mais intuitivo que C/C++ e possibilitar simulação com rápida visualização para vários algoritmos. Também existem vários algoritmos já implementados como, por exemplo, multiplicação de matrizes, FFT e resposta ao degrau, entre outros. Algumas desvantagens são também comentadas, como por exemplo, a dificuldade no tratamento das variáveis, principalmente para não gerarem um *hardware* ineficiente e o fato da simulação no Matlab ser lenta, uma vez que se trata de uma linguagem interpretada.

A proposta de Faldar *et al* (2001) de criar uma ferramenta de conversão direta de scripts de Matlab para VHDL é encontrada a partir da versão 2011b do Matlab. No capítulo 4, que fala da ferramenta desenvolvida para simulação, serão feitas considerações adicionais sobre a conversão direta de scripts do Matlab.

Wu e Chen (2011) comentam que o controle de movimento precisa possuir precisão, dinâmica rápida e execução robusta (desempenho robusto). Em especial comentam que os controladores PID não conseguem seguir por contornos complexos ou trabalhar com controle de movimento em sistemas não lineares.

Wu e Chen (2011) comentam sobre as técnicas *Backward*, *Forward* e segurador de ordem zero (ZOH), observando que a última é mais eficiente. Por fim, o sistema proposto foi implementado em uma FPGA usando um emulador de DSP. A quantidade de portas lógicas usadas no projeto não foi indicada.

O artigo de Banati *et al* (2001) relata a implementação de um filtro digital IIR (resposta ao impulso infinito) em FPGA. O método de conversão foi o bilinear e houve uma preocupação com o tempo de processamento da realimentação, já que a mesma demandava várias multiplicações e somas em cascata, o que poderia restringir o desempenho do sistema. Como forma de minimizar esse problema os autores comentam alguns artifícios que os ajudaram a obter os resultados:

- Os multiplicadores foram implementados usando um método de quantização que mantinha a precisão;
- A realimentação foi matematicamente otimizada de forma que duas realimentações locais foram usadas para serem equivalentes a uma realimentação global;
- Os operadores de adição foram implementados com apenas 12 bits, diferentemente dos multiplicadores que possuíam 23 bits mais um de sinal.

No final do artigo, Banati *et al* (2001) comentam que o projeto foi implementado e ocupou 500 unidades lógicas e operava com velocidades de até 66.7 ns de ciclo. Não foi indicado o erro quando comparado a sistemas reais ou simulados.

Dentro das operações com ponto flutuante em FPGA, o artigo de Al-Ashrafy *et al* (2011) comenta sobre a implementação do IEEE 754, que é a norma para implementar o ponto flutuante com precisão simples. Segundo os autores, isso já foi tentado em vários outros artigos. Do ponto de vista desta dissertação, os autores infelizmente não comentam quantas unidades lógicas foram utilizadas para cada operação em ponto flutuante, o que permitiria uma comparação com implementações equivalentes em ponto fixo. Os autores comentam apenas que conseguiram desempenho na ordem de 301 MFLOPs (Mega *FLoating-point Operations Per Second*).

O artigo de Alecsa e Onea (2010) fala sobre a implementação de um controle de velocidade para um motor DC *Brushless*. O artigo é bem completo, apresentando detalhes técnicos de acionamento, simulação e implementação em FPGA. O controle de velocidade é realizado com MOSFETs no acionamento e a realimentação é feita com tacômetros ópticos usando a técnica do pulso em quadratura que, segundo os autores, oferece uma interface digital de fácil montagem com sistemas digitais, como as FPGAs. O controle do sistema foi realizado com o clássico controlador PID, mas usando apenas a parte PI.

No artigo de Alecsa e Onea (2010) a implementação do controlador PI é via equação a diferenças e o controlador foi discretizado usando o método bilinear. Uma vez alcançada à estrutura do controlador a mesma foi implementada no pacote Simulink do Matlab usando as ferramentas integradas da empresa Xilinx (*Xilinx System Generator*). Segundo os autores, o perío o (*clock*) final máximo do sistema

ficou em aproximadamente 20ns. Não foi comentado se o sistema foi implementado de forma paralela ou serial ou quantas unidades lógicas foram utilizadas.

Um ano antes os mesmo autores, Alecsa e Onea (2009), apresentaram um artigo no IEEE sobre a construção de um controlador do tipo DeadBeat, de forma discreta em uma FPGA, para o controle de um motor de corrente contínua. Uma análise interessante dos autores foi sua recomendação sobre as características das placas onde serão implementadas os controladores DeadBeat. Segundo os autores as recomendações são:

- Entre 72 e 648Kbits usados na memória RAM;
- Entre 4 e 36 multiplicadores dedicados;
- Entre 2 e 8 gerenciadores digitais de clock;
- Entre 2 e 33 mil unidades lógicas usadas.

Ainda com Alecsa e Onea (2009), a implementação no pacote Simulink do Matlab é interessante para a simulação de sistemas variantes no tempo como: comunicação, controle, processamento de sinais, processamento de vídeo e processamento de imagem. Além disto, o artigo trouxe várias informações interessantes para a área de implementação de controladores em FPGA, por exemplo:

- O período de amostragem foi escolhido de forma que possibilitasse o controle ter 2 vezes a potência do sistema, isso é conseguido com a escolha de uma amostragem que conseguisse representar qualquer tempo de subida da planta, ou seja, sua maior variação;
- O sistema foi representado por equações a diferenças na FPGA e no Simulink, de forma que sua alteração é rápida tanto na arquitetura quando no tipo de dados (precisão, ponto flutuante ou ponto fixo). Nesse caso, os autores usaram ponto fixo de 18 bits;
- A implementação mais rápida possível vem da realização de todas as operações em paralelo, como por exemplo, multiplicações e somas;
- Os autores usaram um número representado por 18 bits com saturador. Isto aumentou a quantidade de unidades lógicas utilizadas mas impossibilitou um possível e indesejado estouro de memória (*overflow*);
- Segundo os autores, a simulação com o Simulink juntamente com o *System Generator for DSP* trás resultados precisos com relação aos “bits e ciclos”, de

forma que a simulação é exatamente como a implementada. Isto é indicado também por Alecsa e Onea (2009) *apud* [6];

- Quando comparada a simulação dos sistemas implementados com ponto flutuante ou com ponto fixo a diferença se apresentou muito pequena, para os 18 bits utilizados na representação;
- Segundo os autores, uma máquina de estado finita (*finite state machine* - FSM) foi necessária para o controle de cada parte do circuito de controle e a comunicação entre cada uma das três entidades foi feita por um sinalizador (*handshake*) do tipo *ready/strobe*.

Para finalizar, Alecsa e Onea (2009) não indicaram quantas unidade lógicas foram usadas em sua implementação, apontando porém, que todo o ciclo de cálculo era executado em 40ns.

Aproximadamente um ano depois, Alecsa e Onea (2010) publicaram um artigo descrevendo a validação do controle de um motor de corrente contínua (CC) em FPGA. O controle proposto é para um motor CC *Brushless* de baixa potência. O sistema foi testado no Simulink usando o cosimulador do Modelsim com base em HDL.

Para Alecsa e Onea (2010) é comum o uso de DSPs para o processamento de sinais e controle devido a sua alta velocidade. Contudo, quando se usa o paralelismo disponível nas FPGAs, as mesmas se tornam mais rápidas que os DSPs, mesmo com cálculos complexos. Devido a isso FPGAs são bons elementos para a geração de controladores dedicados com estrutura paralela para controle de sistemas rápidos ou que demandem algoritmos complexos.

O PWM usado por Alecsa e Onea (2010) usou 11 bits de precisão, permitindo uma taxa de amostragem de até 24,425KHz. Entretanto a taxa de amostragem efetiva do controlador foi de 1 milissegundo. O controlador PI (PID) foi discretizado usando uma transformação Bilinear para gerar uma equação a diferenças. O cálculo numérico foi implementado com ponto fixo de 19 bits sendo 13 bits usados para representação da fração do número.

No simulink foi usado o sistema *System Generator* da empresa Xilinx para simular o comportamento da plataforma FPGA sobre uma planta em tempo contínuo. A simulação em Alecsa e Onea (2009) e Alecsa e Onea (2010) são parecidas, entretanto o artigo mais novo possui mais elementos de simulação,

conforme ilustrado na figura 14. Esta figura apresenta a simulação do simulink usando blocos customizados da empresa Xilinx para simulação em FPGA.

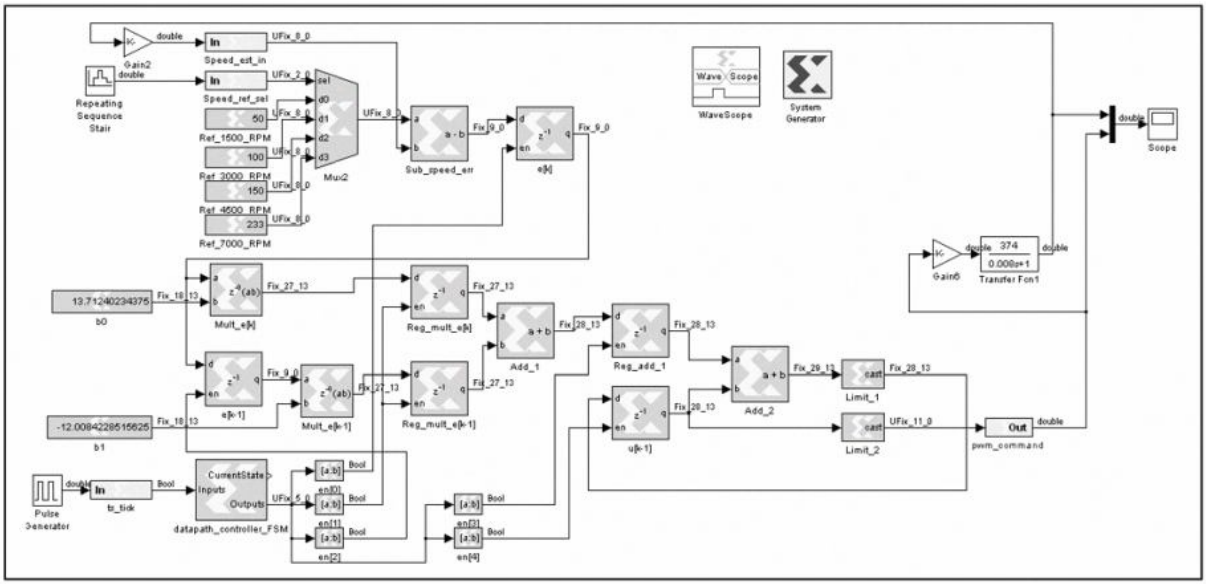


Figura 14 - Integração entre Matlab/Simulink e FPGA da empresa Xilinx
Fonte: Alecsa e Onea (2010, p. 4).

Por fim o artigo de Alecsa e Onea (2009) apresenta a resposta do comportamento do sistema, mas não indica as unidades lógicas, além disso um grande sobressinal é visualizado. A quantidade de unidades lógicas utilizadas com essa abordagem de criação de projetos com System Generator da Xilinx não foi comentado.

O artigo de Lázaro *et al* (2004) fala sobre a implementação de uma rede neural em FPGA. A metodologia é de transferência de algoritmos do Matlab para VHDL automaticamente. O primeiro passo para a implementação é delimitar os limites do algoritmo. No caso do artigo foram usados números inteiros para a entrada de dados e ponto fixo nas operações com isso minimizando a quantidade de unidades lógicas utilizadas. O sistema foi testado em VHDL com a arquitetura *testbench*.

O método de geração de dados foi interessante (LÁZARO *et al*, 2004). Com a confecção de um sistema de teste em *testbench* é possível chamar o programa Modelsim para simular os dados e logo em seguida recebê-los no Matlab. Isso possivelmente agiliza o processo de teste.

Lázaro *et al* (2004) apresentam uma função para calcular o erro entre a simulação em Matlab e os dados alcançado em VHDL. A função de erro é $Erro = \frac{1}{n} * \sum_n \left|\frac{M(n)-V(n)}{M(n)}\right|$, onde M é o número alcançado em Matlab e V é em VHDL. Esse artigo

apresentou uma tabela contendo o total de flip-flops, blocos de memória RAM, multiplicadores de 18x18, etc, somando tudo foi utilizado aproximadamente 28500 unidades lógicas. O sistema conseguiu operar com 50MHz de frequência.

Jastrzebski et al (2005) apresentaram no IEEE um artigo sobre o controle de suspensão magnética de um rotor que foi realizado com a tecnologia FPGA. A implementação da parte numérica foi empírica com 15 bits para as referências, 17 bits para os sensores e realimentação com 14 bits. O controlador foi feito com 528 partes (*slices*) e 2 multiplicadores em hardware. O artigo não comentou sobre a quantidade de unidade lógicas utilizadas, ou se foi realizado algum processamento em paralelo, ou mesmo qual foi o ciclo de controle utilizado.

O artigo de Alonso et al (2010) fala sobre a solução de equações lineares do tipo Ax=B em FPGA. A linguagem de hardware foi o VHDL e a simulação foi com o software Modelsim 6.3f usando o padrão IEEE 754 com ponto flutuante no formato de precisão dupla. Segundo os autores e sua pesquisa sobre algoritmos de solução de equações lineares, a divisão foi à operação matemática que mais despendeu tempo de processamento e necessidade de uma implementação em hardware mais complexa. A arquitetura de paralelismo pode auxiliar na diminuição do tempo de processamento das funções. Da mesma forma, dentro de uma FPGA é possível implementar os processadores, um em paralelo e um em série, e cronometrar os tempo de forma a comparar os reais ganhos nos formatos de implementação.

O artigo de Banerjee *et al* (2003) fala sobre a conversão de ponto flutuante em ponto fixo via Matlab para aplicações em FPGA. Segundo os autores a primeira etapa é o uso de quantizadores existente no toolbox de análise e projeto de filtros (*Filter Design and Analysis* - FDA) do Matlab. Para o caso de uma conversão direta de Matlab para VHDL ou Verilog eles indicam o uso do AccelFPGA que é uma ferramenta de síntese de hardware.

Segundo Banerjee *et al* (2003) o processo de auto-quantização passa por 6 estágios, sendo realizado na seguinte sequências: Escalarização, Nivelamento, Cálculo das faixas alcançadas pelas variáveis, propagação dos limites das variáveis, reestruturação dos limites das variáveis e por fim auto-quantização. Para testar essa metodologia foram selecionados problemas indicados pelos autores como sendo Benchmarks e suas respostas foram apresentadas em uma tabela, para maiores informações recomendamos a leitura do artigo.

Para Leong *et al* (1999) aplicações diferentes requerem precisões matemáticas diferentes para um mesmo grau de precisão de funcionamento. Em seu

artigo sobre processamento de imagem com ponto fixo sua conclusão foi que a conversão automática de ponto flutuante para ponto fixo é um constante obstáculo para o projeto de sistemas de protótipos rápidos. Devido ao erro é necessário um bom conhecimento sobre cálculo aritmético de ponto fixo. A otimização de um sistema com ponto fixo comparado ao de ponto flutuante gera um menor custo de hardware (unidades lógicas) e também menos recursos humanos para implementação.

O artigo de Dufour *et al* (2008) trata de experimentos de controle sobre um motor síncrono implementado em FPGA, ou seja, um simulador em hardware. O modelo do motor foi criado em HDL para simular condições de contorno do motor sem criar danos físicos ao motor real. Os testes foram realizados com o sistema eDRIVEsim que trabalha com plantas mecânicas como automóveis em plataformas de tempo real (*real-time*).

Segundo os autores (DUFOUR *et al*, 2008) o motor virtual usado tem a vantagem de demonstrar facilmente os impactos da variação dos parâmetros do controlador sobre a planta (motor). Entretanto existem dois problemas no sistema a ser simulado que são a saturação do motor e a variação da indução que não são visualizados no modelo gerado.

Hardware-in-the-loop (HIL) é uma técnica desenvolvida para testar sistemas complexos em tempo-real usando modelos matemáticos. Segundo Dufour *et al* (2008) *o* HIL é extensamente usado na indústria e existem algumas vantagens na simulação do motor em FPGA quando comparado a um PC tradicional. De forma resumida elas são:

- Latência baixa entre a entrada e a saída: como o FPGA trabalha com altas velocidades, o atraso (delay) é baixo entre a entrada e a saída;
- Capacidade de inserir proteções diretamente na placa: alguns sistemas de controle não conseguem levar em consideração as proteções eletrônicas necessárias para o acionamento, no caso da FPGA essas proteções podem ser implementadas diretamente na placa;
- Diagrama de blocos codificado diretamente para FPGA: o *Xilix System Generator* pode gerar automaticamente o bloco em uma linguagem HDL de forma que o projetista não precisa ter habilidades avançadas na descrição de hardware, basta conhecer bem o pacote Simulink do Matlab;
- Sistema inteligente de divisão de tarefas: com as FPGAs é possível ter diferentes controladores atuando no sistema como um todo e cada uma com

sua própria taxa de amostragem e controle. Em casos onde é usado um processador padrão IBM-PC é necessário calcular a banda utilizada por cada subsistema para que o processador consiga realizar todos os cálculos a tempo. Sistemas operacionais de tempo real podem auxiliar nesse processo, entretanto ainda é mantida a restrição sobre a quantidade máxima de processadores.

Para Dufour *et al* (2008) existem desafios a serem superado por parte do uso de FPGAs nesses sistemas que são o uso de ponto fixo para os cálculos e a falta de flexibilidade por parte da programação. Uma conclusão interessante dos autores é que a implementação de processamentos em FPGA com baixos tempos de amostragem, como por exemplo, 10 nano segundos, não melhoram automaticamente a precisão dos resultados, isso ocorre devido ao cálculo do ponto fixo usado nas equações. Dessa forma esse recurso é limitado por uma relação custo benefício entre precisão e quantidade de recursos da FPGA que foram usados.

Os sistemas foram testados usando o software CARSIM que recebe os comandos via VehicleSim do Matlab/Simulink. Existe uma versão educacional chamada CarSimEd.

O artigo de Dufour *et al* (2008) mostra os parâmetros do motor pesquisado de forma que é possível gerar seu modelo matemático. Esse artigo não indicou quantas unidades lógicas foram necessárias para o projeto, apenas que o controle e a instrumentação estavam operando a 50 microssegundos.

O artigo de Sreenivasappa e Udaykumar (2009) fala da implementação do controlador PID digital em FPGAs de forma a minimizar as unidades lógicas utilizadas para diminuir diretamente na quantidade de energia consumida.

Segundo os autores o controlador PID é o controlador mais usados dos últimos 50 anos e ele está hoje em aproximadamente 95% dos processos industriais. Devido a não obrigatoriedade de um modelo analítico preciso para o projeto do controlador ele se destaca nas áreas de robótica, automação, controle de processos, manufatura, transporte e aplicações em tempo real multitarefas.

Sreenivasappa e Udaykumar (2009) escolheram a implementação digital por *Backward Difference* e usaram uma metodologia de quatro passos para a implementação em FPGA, tal metodologia é indicada cronologicamente a seguir:

1. Criação e simulação no software Modelsim da empresa Mentor Graphics;

2. Síntese com o Quartus II da empresa Altera;
3. Montagem dos diagramas na forma de esquemático;
4. Geração do arquivo final para ser baixado na placa da FPGA;

Por fim Sreenivasappa e Udaykumar (2009) comparam duas placas das empresas Altera e Xilix e comparam a quantidade de energia gasta para cada uma (miliwatts), mas não comentam se foram utilizados números inteiros, ponto flutuante ou ponto fixo. O período alcançado de processamento indicado pelos autores foi de 13,2 nano segundos (ns).

O artigo de Qu *et al* (2008) propõem a implementação de um otimizador assíncrono por enxame de particular (APSO - *asynchronous particle swarm optimization*) para achar os parâmetros de um controlador PID, entretanto a malha é fechada com um programa gerado pelo programa DSP builder e testado no programa Modelsim antes de ser efetivamente compilado no programa Quartus II da empresa Altera. Testes iniciais também foram executados com o simulink do pacote Matlab.

Para analisar a função de *fitness* Qu *et al* (2008) utilizaram o índice ITAE (integral do erro absoluto multiplicado pelo tempo) comumente usado em controle, entretanto foi implementado um ITAE discreto e não o contínuo, ou seja, o período de amostragem foi levando em consideração.

Devido ao grau de complexidade e da quantidade de itens foi necessário gerar uma máquina de estados com 9 estados para o controle dos passos entre procurar, guardar e aplicar os melhores indivíduos selecionados com o APSO.

Segundo Qu *et al* (2008) e sua metodologia os passos para concepção, teste e compilação em HDL são:

1. Mediante o software DSP builder, o VHDL deve ser carregado no Simulink;
2. O arquivo do Simulink (.mdl) deve ser convertido para VHDL após testes iniciais e em seguida deve ser gerado com TCL (*tool command language*) para fazer com que o Modelsim execute uma sequência de comandos;
3. Parametrização dos arquivos VHDL com base no script do TCL;
4. Execução de um *testebench* no Modelsim e análise os resultados.

Segundo Qu *et al* (2008) o sistema numérico foi expresso usando o SBF que consiste de 8 bits para a parte inteira e 8 para a parte fracionária. Com essa base de

cálculo os resultados foram basicamente os mesmo do Matlab usando sua base com ponto flutuante.

Com base nos gráficos de Qu *et al* (2008) é entendido que o comportamento da planta estava muito bom o que comprova que o controlador estabilizou a planta, entretanto os autores não comparam seu resultado com um caso ótimo ou subótimo com a simulação do Matlab para mostrar o quão próximo do ótimo estavam seus parâmetros. Outro elemento faltante foi o *clock* de operação do sistema (amostragem) e quantas unidades lógicas foram utilizadas nesse projeto.

Um trabalho parecido com o de Qu *et al* (2008) foi o de Chen e Wu (2011) onde a parametrização das constantes PID usando algoritmo genético é proposta com sua realização dentro de uma FPGA via VHDL. A diferença fica na criação do programa via Quartus II da empresa Altera e que as memórias foram criadas usando a ferramenta de propriedade intelectual (IP *core generation tool)* que é a ferramenta que trabalha com blocos prontos para algumas aplicações (IP - *Intellectual Property*). São funções criadas e que não podem ser abertas, ou seja, o usuário as usa como sendo uma “caixa preta”. Novamente, este foi um artigo onde o tempo de processamento ou a quantidade de unidas lógicas não foram comentadas pelos autores.

Um uso nobre para a qualidade de controle dentro das FPGAs é indicado no artigo de Yau *et al* (2005) que fala sobre a implementação em FPGA de uma técnica usada em interpolação ou controle de máquinas de comando numérico computadorizado (CNC) chamado *non-uniform rational B-spline* (NURBS) em tempo real. Segundo os autores o tempo computacional aumenta conforme existe o aumento do grau de precisão do NURBS e na quantidade de eixos de movimento em uma máquina CNC.

Segundo os autores usar um PC ou DSP para o controle de movimento é difícil uma vez que é necessário trabalhar com velocidade entre amostragens de milissegundos, fato que é viável em uma FPGA. O artigo apresenta o modelo matemático teórico (apenas constantes) de um servomotor a ser controlado, o sistema foi discretizado usando o modelo bilinear, entretanto não foi indicado o motivo dessa técnica.

Yau *et al* (2005) fazem uma lista dos principais elementos de controle que fazem com que a quantidade de unidades lógicas variem significativamente, são elas:

- O número de eixos a serem movimentados ao mesmo tempo;

- A quantidade de bits de dados (precisão);
- O grau da curva gerado pelo NURBS;
- O número de malhas fechadas para controlar o sistema;
- O filtro IIR usado, onde ele e o item anterior são fixos para sistemas CNC.

Nesse trabalho de Yau *et al* (2005) os dados usados possuem tamanhos de 16 e 32 bits e o período de amostragem utilizado foi de 0.005 segundos, entretanto a quantidade de unidades lógicas utilizadas não foi comentada bem como as comparações com sistemas reais ou simulados não foram apresentados, ou seja, sem indicação de erro entre os valores alcançados e outros.

No trabalho de Takamatsu *et al* (2006) um controlador do tipo DeadBeat foi implementado em FPGA usando uma multi amostragem paralela baseada na transformação quasi-dq. Segundo os autores existem dois métodos de se empregar o controlador DeadBeat, nesse artigo foi criado um terceiro e comparado com os 2 existentes, sendo considerado pelos autores esse novo método melhor, contendo maior precisão e robustez quando comparado aos anteriores. O tempo entre os cálculos foi de 1,27μs (76 *clocks*), entretanto a quantidade de unidades lógicas não foi apresentada.

Durante a procura por artigos relacionados com essa dissertação, o trabalho de Tao *et al* (2008) foi um divisor de águas no controle com FPGA. O artigo apresenta a implantação de um controle CNC em uma FPGA. Segundo os autores a FPGA se tornou popular nas aplicações de projeto de hardware por prover uma alta performance para processamento digital de sinais, prover soluções 10 ou 100 vezes mais veloz que o habitual quando comparado a um PC ou processador em placa simples (*single board computer* - SBC) e as FPGAs possuem uma arquitetura paralela que é naturalmente aplicado à computação de alta velocidade.

Os chips de FPGA consistem de blocos de memórias, centenas ou milhares, ordenadas em tabelas, conhecidas como *Look-Up Tables* (LUT) que podem ser organizadas para o projeto de forma a criar a solução de processamento necessária. A forma de implementar o LUT é mediante o uso de aritmética distribuída (*distributed Arithmetic* - DA) que é muito promissor em algoritmos e controle (TAO *et al*, 2008).

No trabalho de Tao *et al* (2008) para um deslocamento de trabalho de 30 metros por minuto (30m/min) o ciclo de controle (*sample time*) deve ser de 1ms, ou seja frequência de 1KHz, dessa forma a resolução (precisão) do controlador sobre a

máquina CNC é de 0.5mm e caso a frequência de cálculo seja elevada para 10KHz a precisão melhora para 0.05mm.

No artigo existe uma interessante tabela que demonstra os erros alcançados com base na distância a ser percorrida e no ciclo de processamento do sinal de controle. Com base nesse estudo quanto menor é o tempo de ciclo de processamento menor é o erro de trabalho da máquina. A figura 15 apresenta esse aumento de precisão com o aumento da velocidade.

Position control cycle tcs(ms)	Cycles/s (1/tcs)	Cycle distance with F=3 m/min (mm)	Cycle distance with F=10m/min (mm)	Cycle distance with F=30 m/min (mm)
20	50	1	3.33	10
10	100	0.5	1.66	5
3	333	0.15	0.5	1.5
1	1000	0.05	0.16	0.5
0.4	2500	0.02	0.06	0.2
0.1	10,000	0.005	0.016	0.05

Figura 15 - Relação entre os ciclos de controle e precisão de uma máquina CNC onde o controle é feito via FPGA
Fonte: Tao *et al*, (2008, p. 3).

O controlador utilizado em Tao *et al* (2008) foi o PID e ele foi implementado no FPGA com equação a diferenças. Os testes contemplaram o uso do software Signaltap II da altera para fazer o *debug* do *hardware* de controle (*Hardware in loop* - HIL).

O controle global da máquina CNC usado em Tao *et al* (2008) contém 4 elementos (controladores), 1 controlador PID para a posição, um controle proporcional FF1 mais um proporcional derivativo chamado FF2 e um derivativo de 2º ordem chamado FF3, ou seja uma arquitetura com 6 parâmetros de controle, entretanto não foi comentado se os parâmetros usados foram implementados numericamente como números inteiros, com ponto fixo ou ponto flutuante. Ainda assim, existe uma tabela comparativa sobre os métodos de design e de hardware utilizados. Nessa tabela os autores indicam que a primeira implementação do controlador usou 160 unidades lógicas, a segunda 290 e a terceira 9706.

As arquiteturas se resumem a:

- PCII: arquitetura paralela de integrador (I) mais o elemento PD + FF[(I) + (PD + FFs)] mais 2 blocos de memória da placa;

- PCI: arquitetura paralela de três elementos, sendo eles proporcional e derivativo mais integral que são somados a FFs [(PD) + (I) + (FFs)] mais três blocos de memória da placa;
- Base multiplicadora: todas as memórias usadas são criadas com elementos lógicos, ou seja, sem o uso de memórias externas que diminuiriam a velocidade dos cálculos.

Várias placas com FPGAs encontradas no mercado possuem já interligadas memórias do tipo RAM (*Random Access Memory* – Memória de acesso aleatório) para auxiliar nos projetos e minimizar a quantidade de elementos lógicos utilizados, nesse sentido os autores tentaram otimizar o uso de memórias para minimizar a quantidade de unidades lógicas necessárias no projeto. O ganho na utilização exclusiva de unidades lógicas está na velocidade de trabalho, a arquitetura PCII usou 64 ciclos de máquinas e a PCI usou 33, já na implementação com apenas unidades lógicas, sem usar memória RAM externa, o ciclo foi de 1 único ciclo de máquina.

A conclusão direta do trabalho de Tao *et al.* (2008) é que o uso de memórias externas diminui a velocidade máxima de trabalho e essa velocidade traz impacto no grau de precisão da máquina.

No artigo de Chan *et al* (2004) eles afirmam que o projeto de controladores PID em FPGA usando a técnica da aritmética distribuída minimiza o uso de unidades lógicas em até 13% dos valores tradicionais, além disso o consumo de energia é reduzido em 40%. Os atrativos de se trabalhar com controladores discretos em FPGA segundo os autores são:

- Cálculos de alta velocidade;
- Funcionalidades complexas;
- Capacidade de processamento em tempo real;
- Baixo consumo de energia;

A soma desses itens transforma a FPGA em um elemento atrativo para uso em sistemas embarcados. O controlador PID foi implementado com equação a diferenças. O estudo apresenta comparativos interessantes para sistemas de controle críticos como, por exemplo, ciclos de máquina e a quantidade de unidades lógicas e blocos de memória usadas, entretanto a base numérica não foi indicada e uma análise comparativa entre essa técnica nova e seu erro quando comparado

com técnica tradicional não é apresentada, o que demanda um estudo para a sua comparação (CHAN et al, 2004).

Uma comparação interessante sobre controladores implementados em FPGAs e os DSPs foi apresentado em Idkhajine *et al* (2010) onde o filtro de Kalman estendido (*Extended Kalman Filter* - EKF) foi usado para controlar a velocidade e corrente de um motor de corrente alternada (CA). A conclusão do artigo é que a resposta das FPGAs é melhor, entretanto não foi comentado quantas unidades lógicas foram utilizadas, mas um dado importante que demonstra a precisão da FPGA foi a grande diferença no tempo de execução que impactou no período de amostragem dos controladores. No caso com FPGA o sistema teve uma execução de cálculos com 6 microssegundos enquanto que com a DSP esse tempo foi aproximadamente 10 vezes maior, sendo indicado pelo autor como sendo de 66 microssegundos. A base numérica usada em FPGA foi o ponto fixo com 13 bits e o clock usado foi de 50MHz.

Um trabalho envolvendo controladores em FPGA para exoesqueleto foi apresentado por Lee e Jung (2006). Os autores apresentaram um estudo sobre a relação de robôs mestres e escravos onde o robô mestre é “dirigido” (movimentado) por um humano como se fosse um exoesqueleto e o robô escravo copia os movimentos e os executa de forma igual. Segundo os autores foram usados 12 controladores PID para um total de 11 juntas mecânicas.

Segundo Lee e Jung (2006) foram colocados 12 controladores PID dentro de uma FPGA de 11.520 unidades lógicas. Um relato importante dos autores é que a multiplicação usando elementos seriais é uma desvantagem quando comparado com multiplicadores em paralelo. Isso indica que é uma vantagem se aproveitar do paralelismo real intrínseco das FPGAs para realizar os cálculo em paralelo e ganhar em desempenho. Apenas para exemplificar, segundo os autores, a multiplicação em paralelo para um controlador PID usou 2 ciclos de maquina enquanto usando a arquitetura serial foram necessários 18 ciclos.

Algumas conclusões importantes apresentadas por Lee e Jung (2006) são:

- A FPGA não possui cálculo com ponto flutuante: isso limita a colocação de parâmetros do controlador PID a números inteiros ao invés de números reais;
- O ruído observado foi minimizado com o aterramento adequado dos cabos, ainda assim como os cabos estavam próximos havia casos de dados corrompidos por interferência mútua;

- Como as dimensões mecânicas do robô mestre e escravo são diferentes, isso causa comportamentos mecânicos diferentes e por consequência o seguimento de trajetória não é perfeito.

Para finalizar a revisão Lee e Jung (2006) não comentam quantas unidades lógicas foram usadas por controlador, não indicaram o erro percentual quando comparada a saída e a referência, também não comentaram sobre os sensores e atuadores utilizados.

Segundo o artigo de Urriza *et al* (2009) a conversão dos algoritmos de simulação em Matlab para DSP com ponto fixo consome muito tempo. Além disso, o ponto fixo e o ponto flutuante são de difícil implementação na FPGA. Para os autores, descrever o controlador em VHDL permite uma alta flexibilidade e independência tecnológica quando comparado ao antigo método de gerar um código para implementar em um DSP. As simulações foram realizadas com o Advance MS da Mentor Graphics. Os autores não indicaram a quantidade de unidades lógicas utilizadas.

Após essa revisão bibliográfica sobre o estado da arte é apresentado o próximo capítulo onde é descrita a metodologia adotada nessa dissertação.

4. METODOLOGIA ADOTADA

A metodologia do trabalho faz uso das duas principais áreas de estudo exploradas na introdução desse trabalho: controle de sistemas em tempo discreto e síntese de *hardware* em FPGAs.

É possível organizar a metodologia em cinco etapas, descritas em ordem de execução:

1. Identificar o modelo matemático da planta e achar a arquitetura de controle mais adequada e precisa para a mesma;
2. Implementar o controlador escolhido e/ou a planta em Matlab. O objetivo é achar sua precisão subótima (SILVA FILHO, 2000), ou seja, aquela que não é a ótima, mas que gera uma resposta que atenda as especificações de projeto;
3. Implementar a planta e/ou controlador em VHDL usando o ponto fixo com a mesma precisão utilizada no Matlab;
4. Gerar o *hardware* em FPGA e simular o comportamento com a ferramenta *waveform* (ALTERA, 2012). É gerado um arquivo de dados a ser carregado no Matlab;
5. Por último, comparar os resultados gerados em Matlab e na simulação de *hardware* e ver se existe alguma discrepância estática ou dinâmica, como um sobre sinal ou erro em regime permanente.

A figura 16 apresenta de forma gráfica as etapas descritas. Em destaque está a ordem principal. Com essa metodologia é possível simular apenas as plantas ou as plantas com os controladores em hardware.

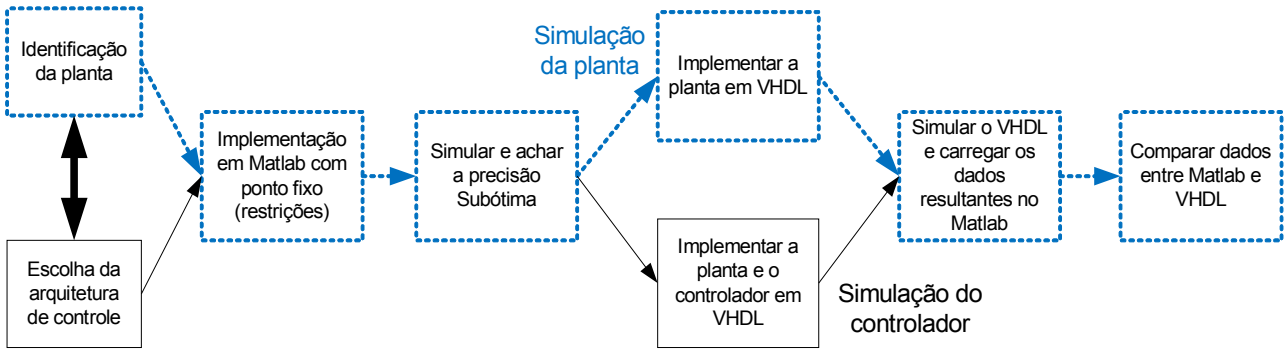


Figura 16 – Apresentação das etapas para simulação do comportamento da planta para um controlador confeccionado em ponto fixo e com limitação de casas de precisão
Fonte: O Autor.

Em seguida, as duas principais ferramentas de software, o Matlab e Quartus II, são discutidas com maior riqueza de detalhes.

4.1 FERRAMENTA DE SIMULAÇÃO EM SOFTWARE

A ferramenta desenvolvida em Matlab/Simulink trabalha com vários parâmetros:

1. Parâmetros da planta: são os pólos e zeros que definem a representação discreta da planta;
2. Taxa de amostragem: é o período entre cada amostra dos sinais de realimentação ou entre cada cálculo da saída dos diferentes blocos do sistema;
3. Estrutura e constantes dos controladores: são os pólos e zeros que definem a representação discreta do controlador, bem como sua topologia;
4. Precisão dos números: O conjunto de diferentes combinações de números de bits antes e depois do ponto permite analisar o impacto da precisão na resposta do sistema. Este parâmetro é crucial e está diretamente relacionado à realização do ponto fixo na FPGA.

Após os parâmetros terem sido indicados, o Matlab/Simulink simula, para cada caso de precisão numérica considerado, apenas a planta ou o conjunto planta mais controlador.

A versão do conjunto Matlab + Simulink utilizada foi a 2011b. O autor não recomenda o uso do Simulink 2009b, pois este apresenta alguns erros de simulação com sistemas rápidos como, por exemplo, servomotores. Além disto, o ponto fixo, como opção de operação numérica, só está disponível a partir da versão 2010.

A arquitetura dentro do software Simulink é indicada na figura 17. Os elementos básicos de controle são mantidos, como por exemplo, referência, controlador, planta, sensor e saída. Em especial, essa arquitetura proposta permite a inserção de um sinal de ruído no sensor. Para tanto, soma-se ao sinal do sensor um ruído branco mais um sinal senoidal com amplitude variável. Nessa dissertação não será usado nenhum padrão de ruído nos testes, pois este bloco adicional em FPGA introduziria mais uma variável para análise dos resultados, tornado esta análise do sistema mais complexa. Futuramente testes com ruído e perturbações podem ser considerados, a exemplo do artigo de Oliveira, Lima e Lopes (2011) sobre

controladores PID sintonizados com a técnica de otimização por enxame de partículas (PSO – P*article Swarm Optimization*).

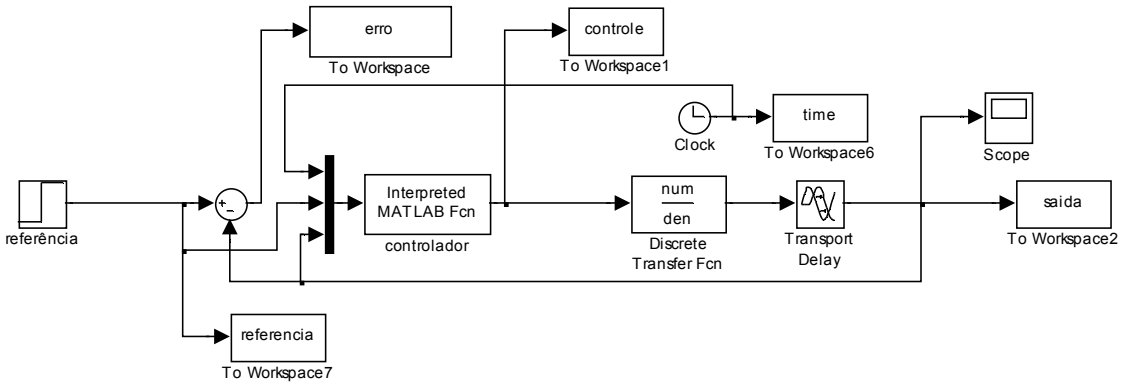


Figura 17 - Diagrama de simulação no pacote Simulink do software Matlab para simulação de controladores com várias faixas de precisão
Fonte: O Autor.

O bloco controlador indicado na figura 17 é descrito de forma textual, usando um *script* (arquivo com extensão .m). Desta forma, várias arquiteturas alternativas podem ser criadas e testadas.

A visualização global das etapas internas do controlador confeccionado em Matlab/Simulink são apresentadas:

- Etapa um e mais externa: nela são indicados os parâmetros do controlador, taxa de amostragem e precisão dos cálculos com ponto fixo em linha de comando do Matlab. Este procedimento permite a futura utilização do *script* de forma genérica.
- Etapa dois e central: o arquivo em Simulink é chamado em linha de comando do Matlab. Internamente, o *script* do controlador e outros blocos associados são executados, gerando os vetores de erro, de controle e de saída;
- Etapa três e final: trata-se da execução do próprio *script* do controlador. A cada nova interação (*loop*) este *script* é executado, gerando o sinal de controle que será utilizado pelos outros blocos do arquivo simulink. Esse sinal de controle é usado externamente como um sinal do tipo *Double*. Isto ocorre uma vez que o Simulink, em suas funções de transferência e de forma padrão, utiliza esta formatação de dados. Contudo, internamente os cálculos são mantidos com a precisão indicada em ponto fixo.

A figura 18 representa de forma gráfica as etapas descritas.

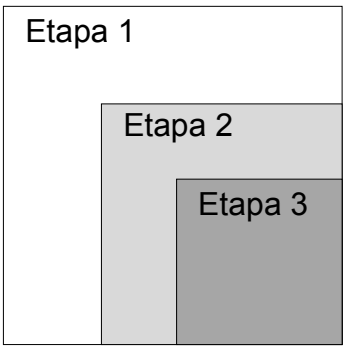


Figura 18 – Apresentação das etapas para simulação do controlador confeccionado em ponto fixo e com limitação de casas de precisão
Fonte: O Autor.

Na figura 19, é apresentado um exemplo do *script* do controlador criado para simulação. No exemplo é necessário trabalhar com as seguintes variáveis:

- Dados de entrada no tempo: tempo, sinal da planta e sinal da referência;
- Constantes do controlador: Kp, Ki, Kd do controlador PID;
- Variáveis auxiliares para contas: são o erro e o sinal intermediário chamado aux;
- Memória de estado: são as indicações de erro e sinal de controle nas interações anteriores. Eles são necessários na função do controlador PID (exemplo: e(k-1) e u(k-2));

```
function ut = controlador_pid(ent)
global uut ut_1 e e_1 e_2 dados
global kp ki kd dt rt yt yy xx
tnow = ent(1);
rt   = ent(2);
yt   = ent(3);
if abs(tnow) <= eps
    disp(' ')
    disp(' inicializando ... ')
    clear dados
    ut_1 = 0;
    disp(' ok! ')
end
e=rt-yt;
aux=(ut_1+e*(kp+(ki*dt)+kd/dt)+e_1*(-kp-2*kd/dt) +e_2*kd/dt);
ut=double(aux);
ut_1 = aux;
e_2  =e_1;
e_1  =e;
end
```

Figura 19 – Exemplo de função de controlador criado como script para simular o comportamento de um controlador com limitação de casas decimais de precisão
Fonte: O Autor.

A ferramenta proposta permite a análise quantitativa e qualitativa dos dados. Os dados quantitativos e qualitativos foram separados da seguinte maneira:

- Quantitativos: números alcançados e usados como métricas comparativas e indicativas de qualidade do projeto. Por exemplo: o índice IAE (*Integral of Absolute Error*) descreve a soma do valor absoluto do erro entre a referência e a saída da planta (erro). Outro exemplo é o erro de porcentagem entre os valores dos coeficientes do controlador, com e sem restrição de precisão.
- Qualitativo: o principal índice qualitativo analisado foi o sobressinal máximo alcançado. Este elemento é importante, pois é uma especificação de desempenho de projeto e como tal, deve ser respeitado. Outro índice qualitativo é o ponto de flexão onde o sistema começa a ser, ao menos, marginalmente estável, conhecido na literatura como tempo de acomodação do sistema. Por último, um item qualitativo importante de verificação é o erro em regime permanente.

Um exemplo de análise quantitativa é apresentado na tabela 3. É possível observar os índices de erro alcançados para cada representação em ponto fixo.

Essa tabela demonstra o comportamento da planta para cada configuração de precisão nos cálculos do controlador, ou seja, permite uma análise qualitativa sobre a precisão mínima necessária para que o controlador funcione adequadamente, atendendo as especificações de projeto. As constantes da tabela 3 são:

- PP: erro entre o valor real da constante Kp do controlador quando comparado ao valor implementado em FPGA;
- PI: erro entre o valor real da constante Ki do controlador quando comparado ao valor implementado em FPGA;
- PD: erro entre o valor real da constante Kd do controlador quando comparado ao valor implementado em FPGA;
- Arredondamento: descreve a quantidade de bits utilizados na parte inteira e na parte fracionária dos números, por exemplo, 16:10 quer dizer que foram usados 16 bits para a parte inteira e 10 bits para representar a parte fracionária;
- IAE: é a some do erro absoluto encontrado entre a referência do sistema e o sinal de saída da planta (servomotor com carga);
- U: é a soma do sinal de controle utilizado no experimento;
- SS: é o sobresinal máximo encontrado. Este é um item de projeto recorrente;

- Média dos erros: é a média dos erros entre os parâmetros do controlador PID que são Kp, Ki e Kd.

No exemplo, o modelo apresentado é de um servomotor com um controlador PI.

O teste número 15, destacado na tabela 3, descreve que o índice de erro (IAE) está em 0.846, que à média dos erros absolutos dos parâmetros PP e PI (Kp e Ki, respectivamente) é de aproximadamente 10% e, principalmente, que o sobre sinal não foi superior a 10%, ficando na casa dos 8.3%. Ressalta-se este último dado pelo fato do mesmo ser uma especificação de desempenho do sistema.

Tabela 3 – Exemplo de análise dos dados de simulação para o caso servomotor.

Nº	Controlador	Método	PP(%)	PI(%)	PD(%)	Arredondamento	IAE	U	SS(%)	Média dos erros %
1	PI	serial	100.0000	100.0000	0.0000	16:0	50.017	0.00E+00	0.000	100.00
2	PI	serial	100.0000	100.0000	0.0000	16:1	50.017	0.00E+00	0.000	100.00
3	PI	serial	100.0000	100.0000	0.0000	16:2	50.017	0.00E+00	0.000	100.00
4	PI	serial	100.0000	100.0000	0.0000	16:3	50.017	0.00E+00	0.000	100.00
5	PI	serial	100.0000	100.0000	0.0000	16:4	50.017	0.00E+00	0.000	100.00
6	PI	serial	100.0000	100.0000	0.0000	16:5	50.017	0.00E+00	0.000	100.00
7	PI	serial	100.0000	100.0000	0.0000	16:6	50.017	0.00E+00	0.000	100.00
8	PI	serial	100.0000	100.0000	0.0000	16:7	50.017	0.00E+00	0.000	100.00
9	PI	serial	100.0000	100.0000	0.0000	16:8	50.017	0.00E+00	0.000	100.00
10	PI	serial	100.0000	100.0000	0.0000	16:9	50.017	0.00E+00	0.000	100.00
11	PI	serial	100.0000	100.0000	0.0000	16:10	50.017	0.00E+00	0.000	100.00
12	PI	serial	-9.2596	100.0000	0.0000	16:11	3.843	9.16E-07	-24.711	54.63
13	PI	serial	-9.2596	-53.9639	0.0000	16:12	0.765	1.64E-06	11.706	31.61
14	PI	serial	-9.260	23.018	0.000	16:13	0.812	1.55E-06	0.074	16.14
15	PI	serial	4.398	-15.473	0.000	16:14	0.846	1.58E-06	8.364	9.94
16	PI	serial	-2.431	3.773	0.000	16:15	0.822	1.56E-06	3.915	3.10
17	PI	serial	0.984	3.773	0.000	16:16	0.840	1.56E-06	4.437	2.38
18	PI	serial	-0.724	-1.039	0.000	16:17	0.826	1.57E-06	4.994	0.88
19	PI	serial	0.130	-1.039	0.000	16:18	0.831	1.57E-06	5.131	0.58
20	PI	serial	0.130	0.164	0.000	16:19	0.832	1.57E-06	4.929	0.15
21	PI	serial	-0.084	0.164	0.000	16:20	0.831	1.57E-06	4.894	0.12
22	PI	serial	0.023	-0.137	0.000	16:21	0.831	1.57E-06	4.962	0.08

Fonte: O Autor

A figura 20 apresenta a resposta encontrada com os parâmetros de controle, definidos no teste 15, sobre a planta indicada.

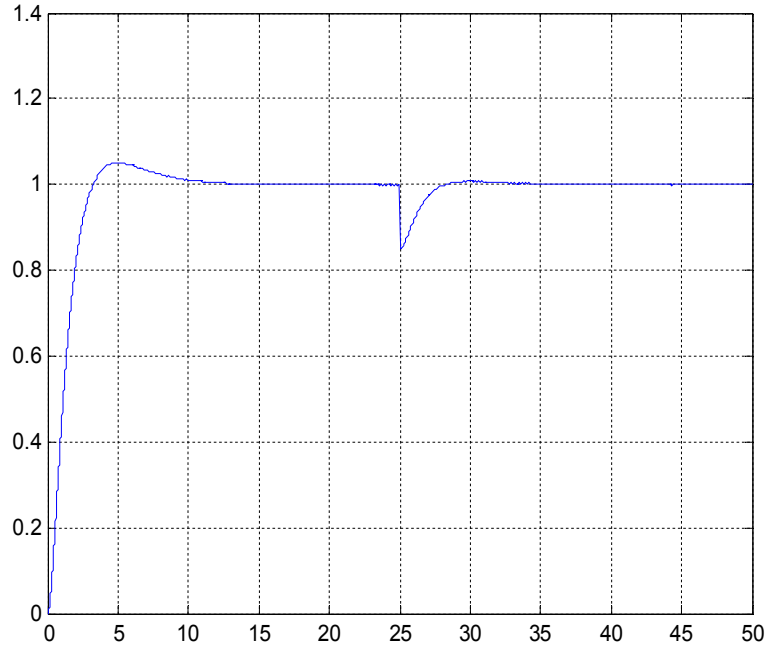
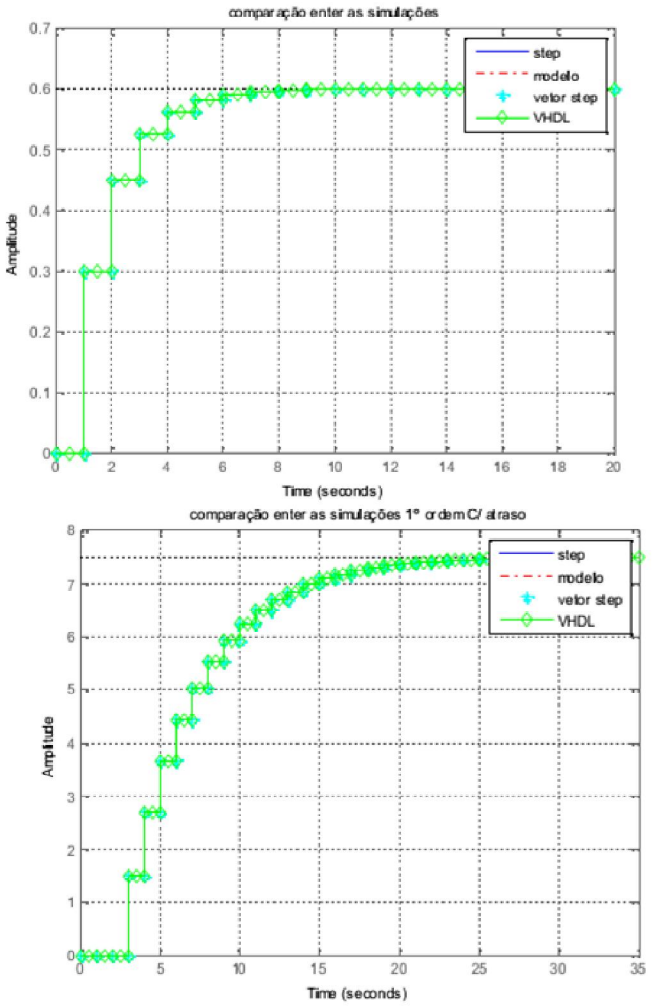


Figura 20 – Apresentação do comportamento do controlador com ponto fixo sobre um servomotor sofrendo com ruídos no sensor e com uma perturbação (carga) aos 25 segundos de simulação. Foi utilizado arredondamento 16:14
Fonte: O Autor.

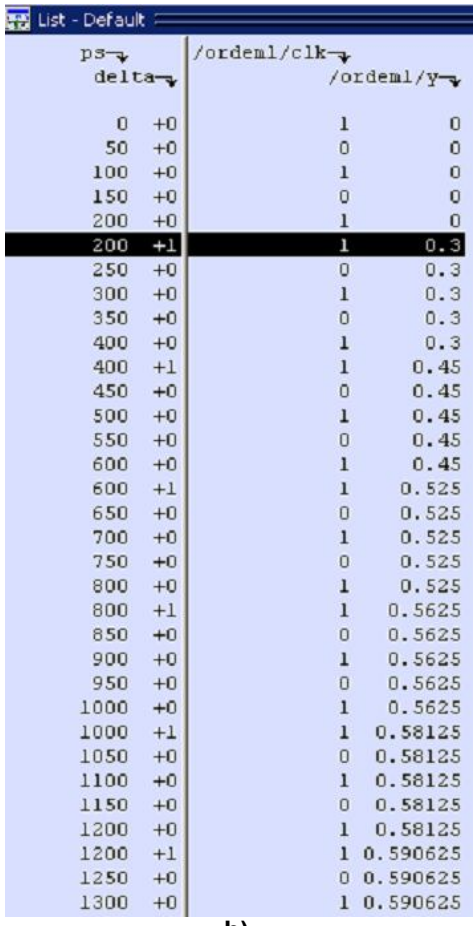
4.2 FERRAMENTAS DE SIMULAÇÃO EM HARDWARE UTILIZADAS

Os controladores PID em *hardware* foram implementados com o software Quartus 9.1 *WebEdition* da empresa Altera. A versão utilizada não é a mais recente, mas sim a última (2009) a integrar a ferramenta de simulação *waveform editor*. Esta ferramenta mostrou-se fundamental no controle dos dados de entrada, saída e variáveis auxiliares.

O software de simulação Modelsim também foi utilizado em testes iniciais para a simulação de modelos matemáticos em *hardware*, mas o mesmo desconsidera limitações de bits e incrementa automaticamente a quantidade de bits utilizada, dessa forma desconsiderando limitações reais de *hardware*. Na figura 21 é apresentada a simulação em *hardware* no Modelsim e a simulação em software no Matlab/Simulink. A simulação em Modelsim está exatamente igual à simulação em Matlab quando comparadas graficamente. Isto pode ser observado na figura 21a, onde as legendas “modelo” e “VHDL” apontam para curvas sobrepostas. Já as curvas relacionadas às legendas “step” e “vetor step” não são representadas no gráfico por motivo de escala. Observa-se, na figura 21b que o Modelsim aumenta automaticamente a precisão dos dados (coluna y), o que descredencia sua aplicação nessa dissertação.



a)



b)

Figura 21 – Apresentação dos dados simulados em Matlab com a resposta alcançada com o simulador Modelsim (versão estudante). a) dois modelos comparados e; b) vetor resposta do Modelsim referente ao gráfico superior da figura 21a, com aumento automático de precisão. Fonte: O Autor.

A empresa Mentor Graphics, que distribui o Modelsim, possui uma ferramenta de simulação de *hardware* em FPGA dentro do Simulink do Matlab, mas essa ferramenta não é distribuída na versão estudante, usada nessa dissertação.

A empresa Altera disponibiliza, além do *waveform editor*, em seu software Quartus 9.1, o Signaltap II, que mostra o comportamento dos sinais internos da FPGA em tempo de execução, ou seja, ele trabalha como um analisador lógico embarcado.

Resumindo, optou-se pela ferramenta *waveform* do Quartus II (pacote de atualização 1 - *sp1*) para gerar os dados de saída da simulação em *hardware*. Estes dados são armazenados em um arquivo de texto e posteriormente usado pelo Matlab para comparação com os modelos simulados em *software*. No item

ferramentas e resultados é explicado o procedimento utilizado para esse processo de comparação dos resultados teóricos e simulados.

Uma observação importante é que o VHDL 2008 com ponto fixo, distribuído por Bishop (2012), é segundo o autor, melhor computado com as regras do VHDL 93. Maiores informações podem ser encontradas no documento original. Em trabalhos futuros é importante no momento da síntese indicar a versão adequadamente.

4.2.1 Descrição da metodologia de controle e simulação em hardware

Para fins de integração dos dados obtidos pelo Quartus II e pelo Matlab, foi gerada uma metodologia de controle e simulação em *hardware* denominada CS8 (controle e simulação em 8 estados). Suas etapas são:

1. Encontrar a função de transferência contínua da planta e do controlador PID. Discretizar a planta e o controlador segundo uma taxa de amostragem necessária e montar as respectivas equações a diferenças;
2. Montar as equações a diferenças no VHDL. Para tanto se utiliza uma máquina de estados síncrona de 4 estados: simulação da planta, atualização das variáveis da planta, geração do sinal de controle, atualização das variáveis do controlador;
3. Sintetizar o arquivo VHDL e verificar se não há erro lógico e se os tempos críticos indicados pelo Quartus II não ultrapassam o período de amostragem utilizado;
4. Simular a resposta no *waveform* com representação binária;
5. Salvar o arquivo no formato tbl (por exemplo, teste6.tbl);
    a. Após salvar o arquivo , abri-lo e limpar o cabeçalho, rodapé e sinal de maior (>) gerado pelo Quartus II;
6. Carregar no Matlab os vetores de *hardware* gerados no formato tbl modificado e caracterizar a precisão a ser usada pelo Matlab durante a geração dos gráficos;
7. Gerar os gráficos no Matlab;
8. Simular no Matlab a resposta do sistema em *software* e comparar graficamente e/ou numericamente (IAE, por exemplo) com a resposta em *hardware* capturada no item 6.

Com base nesse método, o quadro 1 mostra um exemplo de implementação com uma planta de primeira ordem controlada por um controlador PID. Nesse quadro são apresentados os elementos básicos indicados na figura 22, mas aplicados para o conteúdo específico desta dissertação que é a implementação de controladores PID em FPGA usando ponto fixo com o VHDL 2008.

O código do quadro 1 pode ser considerado longo devido à necessidade de variáveis auxiliares para os cálculos e as memórias para armazenamento das constantes e condições passadas dos estados para futuros cálculos. Essas variáveis são necessárias, pois o controlador trabalha com recursividade, ou seja, precisa de valores do sinal do erro passado para calcular o sinal de controle do instante atual.

Depois de vários testes preliminares e análise dos erros de processamento foi descoberto que era necessário gerar um tempo para que os dados calculados fossem devidamente estabilizados. Para gerar esse tempo foi utilizada uma máquina de estados para controlar as condições de cálculo e de armazenamento dos valores em memória, caso contrário à simulação apresenta erros numéricos.

A topologia de projeto, com base na arquitetura básica de VHDL apresentada por Pedroni (2008), é representada pela figura 22:

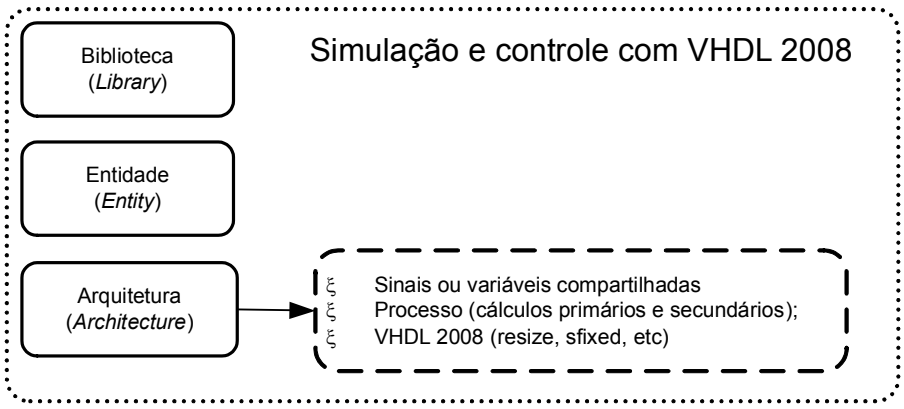


Figura 22 – Apresentação dos itens simplificados do controle e simulação em VHDL 2008
Fonte: O Autor.

Quadro 01 – Exemplo de Programa em VHDL com simulação de controlador PID com máquina de estado em VHDL 2008.

```
library ieee;
library ieee_proposed;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee_proposed.fixed_float_types.all;
use ieee_proposed.fixed_pkg.all;
entity teste17controle is
generic (i:integer:=8;
         f:integer:=23);
```

```
---------- Processo baixo ----------
process (clk)
begin
if (clk'EVENT AND clk='1' and estado=A) then
       estado := B;   end if;
if (clk'EVENT AND clk='1' and estado=B) then
       estado := C;   end if;
if (clk'EVENT AND clk='1' and estado=C) then
       estado := D;   end if;
```

```
  port (
       clk, st : in  bit;
       ref     :  in  integer  range  -1000  to
1000 :=0;
       y       : out sfixed (i downto -f));
end entity teste17controle ;
architecture corpo of teste17controle  is
-----------------------------------------------
TYPE tipo_estado IS (A, B, C, D);
shared variable  estado: tipo_estado := A;
shared variable  yout: sfixed (i downto -f);
------------- Planta -------------------------
signal a1: sfixed (i downto -f);
signal b1: sfixed (i downto -f);
shared variable y1: sfixed (i downto -f);
shared variable y2: sfixed (i downto -f);
signal r: sfixed (i downto -f);
shared variable r1: sfixed (i downto -f);
shared variable aux1: sfixed (i downto -f);
shared variable aux2: sfixed (i downto -f);
------------ Controlador-------------------
signal kp: sfixed (i downto -f);
signal ki: sfixed (i downto -f);
signal kd: sfixed (i downto -f);
signal dt: sfixed (i downto -f);
-----------------------------------------------
shared variable e2: sfixed (i downto -f);
shared variable e1: sfixed (i downto -f);
shared variable e: sfixed (i downto -f);
shared variable u1: sfixed (i downto -f);
shared variable u: sfixed (i downto -f);
shared variable aux10: sfixed (i downto -f);
shared variable aux11: sfixed (i downto -f);
shared variable aux12: sfixed (i downto -f);
shared variable aux13: sfixed (i downto -f);
shared variable aux14: sfixed (i downto -f);
shared variable aux15: sfixed (i downto -f);
shared variable aux16: sfixed (i downto -f);
shared variable aux17: sfixed (i downto -f);
shared variable aux18: sfixed (i downto -f);
begin
r <= to_sfixed (ref, r);
--------- modelo matemático -----------------
b1 <= to_sfixed (0.009662, b1); -- dt=1/1e2
a1 <= to_sfixed (0.9968, a1);   -- dt=1/1e2
------------ Controlador PID -----------------
-----
kp <= to_sfixed (1.9055550, i, -f);
ki <= to_sfixed (0.4755550, i, -f);
kd <= to_sfixed (0.1755550, i, -f);
dt <= to_sfixed (0.01, i, -f);
-- continua na coluna ao lado
```

```
if (clk'EVENT AND clk='1' and estado=D) then
       estado := A;   end if;
end process;
-------------------------
---------- Processo alto ----------
process (clk) -- o processo é usado para
controlar a atualização
begin
if (clk'event and clk='1') then
       CASE estado is
       when A =>
       if (clk='1') then
       aux1 := resize (y1*a1, yout);
       aux2 := resize (u*b1, yout);
       yout := resize (aux1+aux2, yout);
       end if;
       when B =>
       if (clk='1') then
       y1 := yout;
       y <= yout;
       end if;
       when C =>
       if (clk='1') then
       e := resize (r-yout, i, -f);
       aux10 := resize (kd/dt, i, -f);
       aux11 := resize (e2*aux10, i, -f);
       aux12 := resize (kd/dt, i, -f);
       aux13 := resize (kp + 2*aux12, i, -f);
       aux14 := resize (e1*aux13, i, -f);
       aux15 := resize (kd/dt, i, -f);
       aux16 := resize (kp + ki*dt + aux15,
i, -f);
       aux17 := resize (e*aux16 , i, -f);

       aux18 := resize (aux11 - aux14 +
aux17, i, -f);
       u := resize (u1 + aux18 , i, -f);
       end if;
       when D =>
       if (clk='1') then
       e2 := e1;
       e1 := e;
       u1 := u;
       end if;
       end case;
end if;
end process;
end architecture corpo;
```

Fonte: O Autor.

Para o controle entre os cálculos e o armazenamento dos dados nas memórias foi criada uma máquina de estados com 4 estados, descrita na figura 23. Os estados são:

- SA: primeira posição da máquina de estados. Nesse estado é feito o cálculo do comportamento da planta mediante sua equação a diferenças;
- SB: segunda posição da máquina de estados. Nesse estado é feito o armazenamento dos estados da planta em memórias;
- SC: terceira posição da máquina de estados. Nesse estado é feito o cálculo do sinal de controle com base na diferença entre o sinal a planta e da referência desejada;

- SD: quarta e última posição da máquina de estados. Nesse estado é feito o armazenamento dos sinais do controlador e do erro em memórias. Ao final do estado D, a máquina de estados retorna para o estado A;

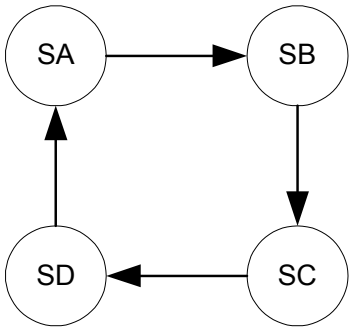


Figura 23 – Máquina de estados para controle dos cálculos e armazenamento nas memórias
Fonte: O Autor.

Na tabela 4 é apresentada a condição de transição dos estados da máquina de estados finitos usado na implementação. A transição é feita pelo clock e sempre é unidirecional.

Tabela 4 – Tabela de transição da máquina de estados referenciada na figura 23.

Estado atual	Condição	Ação
A	Clock = 1	Vá para estado B
B	Clock = 1	Vá para estado C
C	Clock = 1	Vá para estado D
D	Clock = 1	Vá para estado A

4.3 MODELOS UTILIZADOS NOS TESTES

Como o trabalho poderia alcançar conclusões errôneas caso fosse usado apenas uma planta, uma única taxa de amostragem ou mesmo apenas um tipo de controlador, optou-se por testes com algumas plantas suficientemente distintas para minimizar a possibilidade de vícios experimentais. Dessa forma as plantas de estudo são:

- Planta de primeira ordem: essa planta foi retirada do livro de Wilkie, Johnson e Katebi (2002) e representa a dinâmica de um forno de aquecimento. Sua função de transferência é apresentada na equação 19. Essa planta possui um ganho de amplitude 3 e sua constante de tempo (Tal) é de 3,1 segundos.

Com base na teoria clássica de controle esse sistema possui um pólo e nenhum zero.

$$G(s) = \frac{3}{3{,}1s + 1} \tag{19}$$

- Servomotor: os servomotores são equipamentos com dinâmica rápida, com várias malhas de controle e tempos críticos de processamento. O modelo do servomotor utilizado foi retirado do catálogo da empresa Maxon Motors (MAXON, 2011). Esse sistema é de segunda ordem e sua função de transferência é indicada na equação 20. Essa planta possui um ganho de amplitude de aproximadamente 5652, o que é um valor alto. Com base na teoria clássica de controle esse sistema possui dois pólos e nenhum zero.

$$G(s) = \frac{9{,}168 * 10^6}{1622s^2 + 6359s + 1471} \tag{20}$$

- Braço de robô. Esse braço de robô foi indicado no livro de Dorf e Bishop (2001) como sendo um projeto complexo e com grande possibilidade de sobressinal na posição do braço, característica associada ao posicionamento dos pólos e zeros (DORF, BISHOP, 2001, p. 293). Esse sistema é de quinta ordem e sua função de transferência é apresentada na equação 21. Essa planta possui um ganho de amplitude de aproximadamente 1.6. Com base na teoria clássica de controle esse sistema possui cinco pólos e quatro zeros.

$$G(s) = \frac{(s^2 + 4s + 10004)(s^2 + 12s + 90036)}{(s + 10)(s^2 + 2s + 2501)(s^2 + 6s + 22509)} \tag{21}$$

- Pêndulo invertido: o pêndulo invertido é um caso clássico da área de controle, é constituído de uma haste com um peso na extremidade superior e uma dobradiça na parte inferior, esta dobradiça é fixa em um carro que possui motores acoplados a suas rodas, fazendo com que ele se mova para frente e para trás tentando manter o equilíbrio da haste (pêndulo). Esse exemplo foi apresentado no artigo de Bate, McDermid e Nightingale (2003) que trada dos tempos críticos aplicados a sistemas de controle em tempo real. O mesmo modelo é apresentado no site da Universidade Canegie Mellon que trata de controle de sistemas (*Control Tutorials for Matlab*). Esse sistema é de terceira ordem, e é um sistema instável em malha aberta e sua função de

transferência é apresentada na equação 22. Com base na teoria clássica de controle esse sistema possui três pólos e um zero.

$$G(s) = \frac{4{,}546s}{s^3 + 0.182s^2 - 31.182s - 4{,}454)} \tag{22}$$

Para demonstrar as dinâmicas das funções de transferência acima (equações 19 até 22) a figura 24 apresenta as dinâmicas de cada uma das plantas em malha aberta (sem controle) para uma entrada ou excitação degrau unitário.

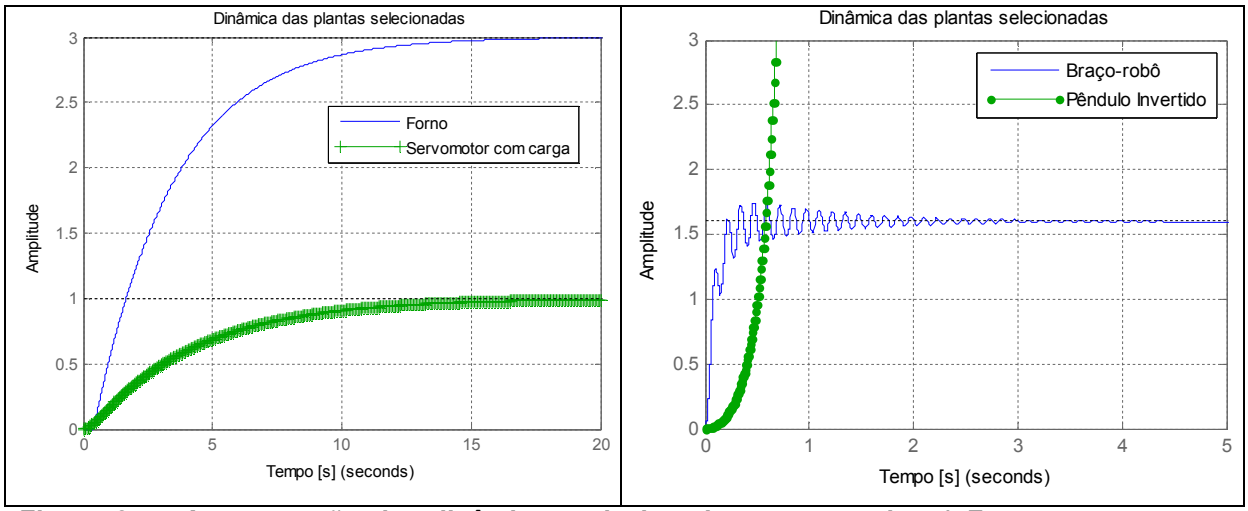


Figura 24 – Apresentação das dinâmicas selecionadas para estudo; a) Forno e servomotor com carga; b) Braço de robô e Pêndulo invertido.
Fonte: O Autor.

Como o Matlab possui várias configurações para realização dos cálculos com ponto fixo, alguns testes preliminares foram realizados na tentativa de determinar o impacto dos parâmetros de configuração nestes cálculos. Tais testes são apresentados no Apêndice B juntamente com a descrição de cada configuração utilizada nessa dissertação.

5. EXPERIMENTOS E RESULTADOS

Nesse capítulo serão apresentados os experimentos, resultados, bem como as condições de contorno utilizadas na realização dos experimentos.

As simulações iniciais para comprovar o funcionamento dos controladores em *hardware* passam por dois tipos de testes:

- Teste estático com constantes na entrada: valores fixos vão ser colocados na entrada do controlador e será coletado o sinal de saída no ciclo de clock seguinte. Isso será realizado com onze valores distintos de entrada;
- Teste de comportamento dinâmico: um controlador e uma planta serão implementados em FPGA e o controlador deverá estabilizar a planta simulada conforme previsto na simulação, ou seja, esses dados serão comparados aos valores alcançados com o Matlab/Simulink.

A equação a diferenças do controlador PID discreto utilizado tanto na FPGA quanto no Simulink é o apresentado na equação 23.

$$u(k) = \left(ut(k-1) + e(k) * (ki * dt) + \frac{kd}{dt} + e(k-1) * \left(-kp - 2 * \frac{kd}{dt}\right) + e(k-2) * \frac{kd}{dt}\right) \quad (23)$$

Alguns testes iniciais foram realizados com a arquitetura do VHDL 2008 para se saber quais são suas dificuldades e respostas a operações simples. Mediante isso, o apêndice C apresenta os resultados encontrados com esses testes. Serão focados os resultados das operações necessárias para os controladores PID, foco da dissertação.

A arquitetura proposta para o controlador PID é indicada no quadro 2. Algumas variáveis auxiliares foram necessárias para minimizar os erros gerados pelo comando *resize* do VHDL 2008. Este comando apresentou resultados inexatos ao calcular a resposta final, quando executando operações conjuntas de multiplicação e soma. Devido a esse fato, as variáveis auxiliares objetivaram as operações de soma e subtração ou multiplicação e divisão em linhas de comando separadas. As constantes PID foram inseridas no corpo do controlador e não como elementos de entrada, pois o software de Simulação do Quartus II não aceita essas entradas com ponto fixo.

O comando *generic* do quadro 2 foi utilizado para deixar o código com maleabilidade na troca da quantidade de bits utilizado, dessa forma gerando uma arquitetura genérica que pode ser utilizada com várias precisões diferentes.

Os elementos de entrada e saída externa no código são: clock do processamento (clk), erro entre sinal de referência e o sinal da planta (ee) e sinal de controle (u), respectivamente. Para esse experimento a taxa de amostragem foi fixa e incorporada ao código do hardware. As bibliotecas usadas são: ieee e ieee_proposed, onde esta última é a biblioteca proposta por Bishop (2012) em seu artigo e também comentada por Pedroni (2012).

Quadro 02 – Exemplo de controlador PID construído com VHDL 2008.

```
- PID com ponto fixo
library ieee;
library ieee_proposed;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee_proposed.fixed_float_types.all;
use ieee_proposed.fixed_pkg.all;
entity teste9pid is
generic (i:integer:=10; -- representação em ponto fixo
        f:integer:=5);
  port (
            clk         : in  bit;
            ee          : in integer range -32750 to 32750 :=0;
            u           : out sfixed (i downto -f));
end entity teste9pid;
architecture corpo of teste9pid is
signal kp: sfixed (i downto -f);
signal ki: sfixed (i downto -f);
signal kd: sfixed (i downto -f);
signal dt: sfixed (i downto -f);
signal e: sfixed (i downto -f);
signal e1: sfixed (i downto -f);
signal e2: sfixed (i downto -f);
signal u1: sfixed (i downto -f);
signal uu: sfixed (i downto -f);
signal a: sfixed (i downto -f);
signal a1: sfixed (i downto -f);
signal a2: sfixed (i downto -f);
signal a3: sfixed (i downto -f);
signal b: sfixed (i downto -f);
signal b1: sfixed (i downto -f);
signal b2: sfixed (i downto -f);
signal c: sfixed (i downto -f);
begin
```

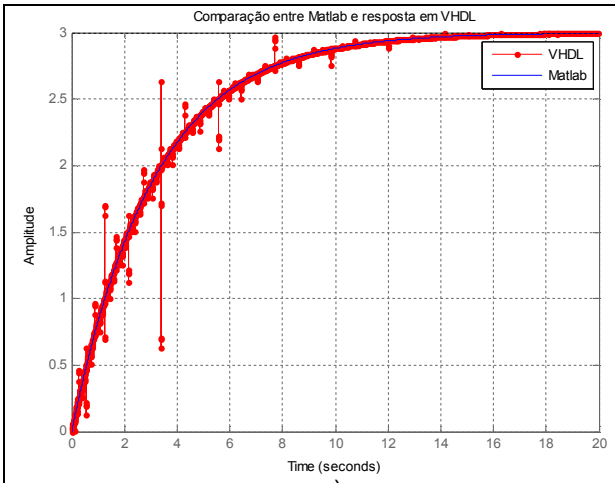
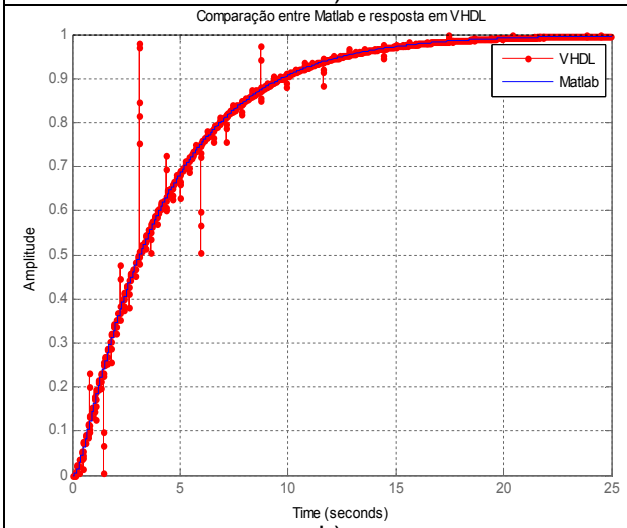
```
-- constantes do controlador PID
kp <= to_sfixed (1.905, kp);
ki <= to_sfixed (0.475, ki);
kd <= to_sfixed (0.175, kd);
dt <= to_sfixed (0.1, dt); -- taxa de amostragem
e <= to_sfixed (ee, dt); -- erro, entrada do controlador

process (clk)
begin
            if (clk'event and clk='1') then
            a1 <=resize(ki*dt, uu);
            ------------------------------------------------------
            --a2 <=resize(kd*reciprocal(dt), uu);
            a2 <=resize(kd/dt, uu);
            ------------------------------------------------------
            a3 <=resize((kp + a1 + a2), uu);
            a <=resize(e*a3, uu);
            ------------------------------------------------------
            --b1 <=resize(2*kd*reciprocal(dt), uu);
            b1 <=resize(2*kd/dt, uu);
            ------------------------------------------------------
            b2 <=resize(kp+b1, uu);
            b <=resize(e1*b2, uu);
            ------------------------------------------------------
            --c <=resize(e2*kd*reciprocal(dt), uu);
            c <=resize(e2*kd/dt, uu);
            ------------------------------------------------------
            uu <=resize(u1 + a - b + c, uu);
            u <= uu;
            e2 <= e1;
            e1 <= e;
            u1 <= uu;
            end if;
end process;
end architecture corpo;
```

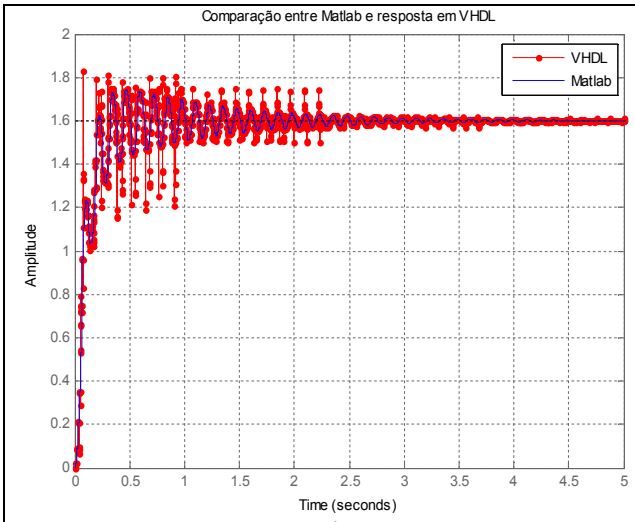
Fonte: O Autor.

5.1 A SIMULAÇÃO EM HARDWARE DAS PLANTAS UTILIZADAS COMO ELEMENTOS DE TESTE

As plantas utilizadas como elementos de teste foram simuladas no *software* Matlab. Estas simulações são usadas como referências nas comparações posteriores com os circuitos equivalentes em VHDL. Nas figuras 25a, 25b, 25c e 25d

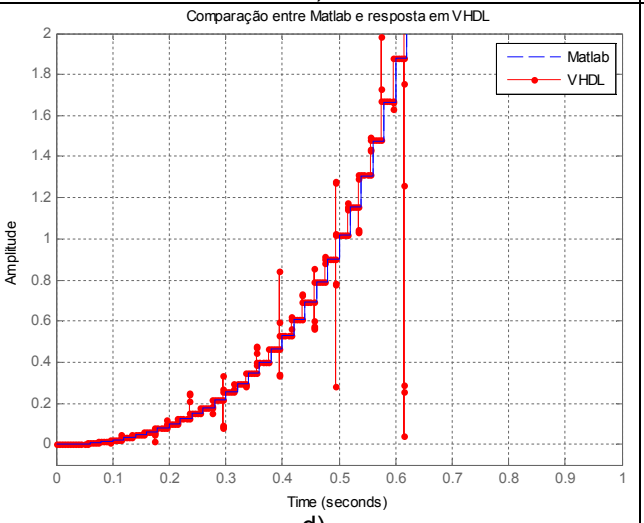
são apresentados exemplos de gráficos comparativos. Nestas figuras, a notação é semelhante à utilizada na tabela 3. Relembrando, [8:23] representa 8 bits associados à parte inteira e 23 bits associados à parte fracionária. IAES (*integral absolute error simulated*) e IAEC (*integral absolute error - circuit*) são as somas dos erros absolutos simulados em Matlab e em circuito (VHDL) respectivamente, quando comparados ao sinal de entrada degrau.

 a)	Planta 1: Forno Bits: 8:23 Amostragem: 0.01segundo IAES: 48.39 IAEC: 54.25 Erro(%): 12.11 Elementos lógicos (EL) utilizados: 466
 b)	Planta 2: Servomotor com carga Bits: 8:23 Amostragem: 0.1 segundo IAES: 1.747 IAEC: 1.901 Erro(%): -8.87 Elementos lógicos (EL) utilizados: 1022



c)

Planta 3: Braço de robô
Bits: 8:23
Amostragem: 0.01 segundo
IAES: 4.074
IAEC: 4.758
Erro(%): -16.79
Elementos lógicos (EL) utilizados: 2974
Observação: sistema naturalmente oscilatório



d)

Planta 4: Pêndulo invertido
Bits: 8:23
Amostragem: 0.02
IAES: 3.065
IAEC: 3.315
Erro(%): -8.16
Elementos lógicos (EL) utilizados: 1281
Observação: sistema naturalmente instável

Figura 25 – Respostas dos circuitos em VHDL 2008 simulando a dinâmica de sistemas físicos. 25a) forno, 25b) servomotor, 25c) braço robótico, 25d) pêndulo invertido
Fonte: O Autor.

Os pontos representados fora da curva, mais conhecidos como *outliers* nas simulações, são resultantes dos períodos de acomodação dos sinais em *hardware*. Esses sinais não são verdadeiramente computados no sistema final, pois estão entre os períodos de amostragem e não sobre eles. Como não existe controle sobre a saída dos sinais simulados pelo *hardware* no Quartus II, o mesmo gera em sua saída o comportamento do circuito montado na escala de tempo dos nano segundos (ns) e com isso são visualizados eventuais *jitters* entre as amostragens.

Na figura 26 é apresentada uma planta com um zoom sobre os *outliers*. Essas variações nos sinais não são aleatórias. Elas acontecem sempre nos mesmos pontos para as mesmas condições de simulação. Caso a amostragem seja alterada, a localização dos *outliers* também muda. Observa-se a tendência de maior número de *outliers* com taxas de amostragens maiores. Evidentemente, para uma mesma

janela de tempo, com taxas de amostragem maiores, um maior número de cálculos é realizado, o que justificaria o maior número de *outliers*. Além disso, em estado estacionário sua ocorrência é diminuída drasticamente, pois a variação do sinal é próxima de zero, ou seja, não ocorrem variações significativas dos sinais nas entradas dos circuitos utilizados para os cálculos.

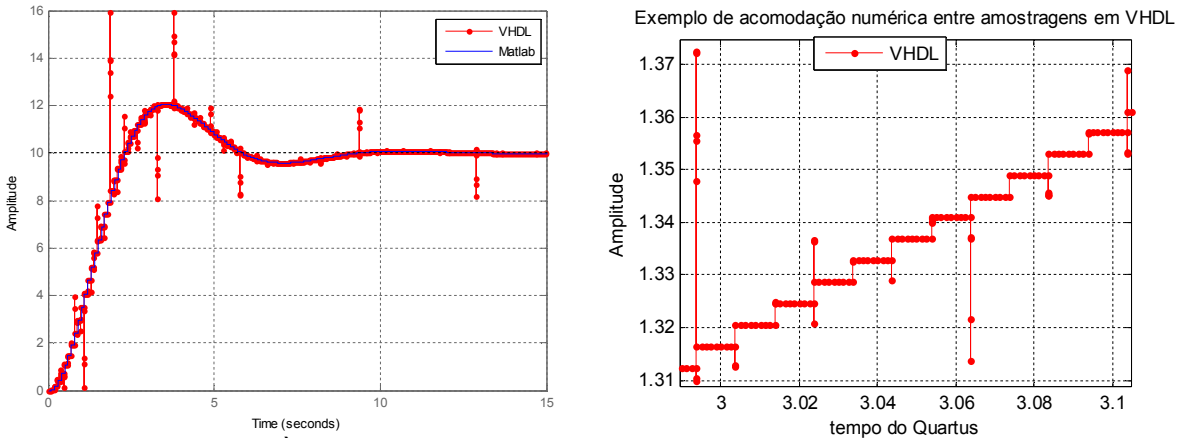


Figura 26 – Apresentação dos outliers. A) vista normal, b) zoom no gráfico
Fonte: O Autor.

Um teste para provar que esses *outliers* não são computados na prática devido a amostragem sincronizada foi possível com a coleta dos dados usando o SignalTap II da altera. O SignalTap II é um analisador lógico e com ele é possível verificar os dados como se fosse um osciloscópio. A figura 27 mostra a comparação entre a simulação do Matlab e o hardware funcionando na FPGA para o caso do servomotor com carga. Conforme é visualizado na figura 27, não existem os *outliers* no hardware no momento da amostragem.

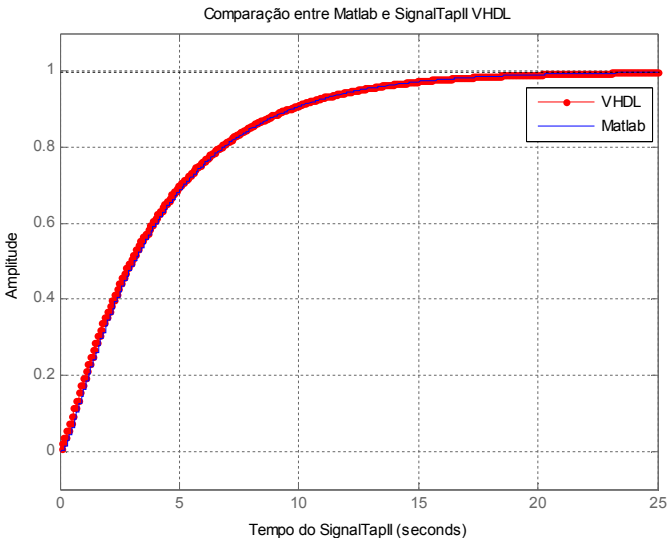


Figura 27 – Dados coletados da FPGA via SignalTAP II. Os dados estão sem os outliers.
Fonte: O Autor.

Para a realização do teste via SignalTap II na FPGA foi adicionado ao código VHDL um divisor de clock para a operação na taxa amostragem correta, uma chave para iniciar o teste e um conjunto de sinais no GPIO (*General Purpose Input/Output*) para coletar os sinais. A tela de teste é apresentada na figura 28.

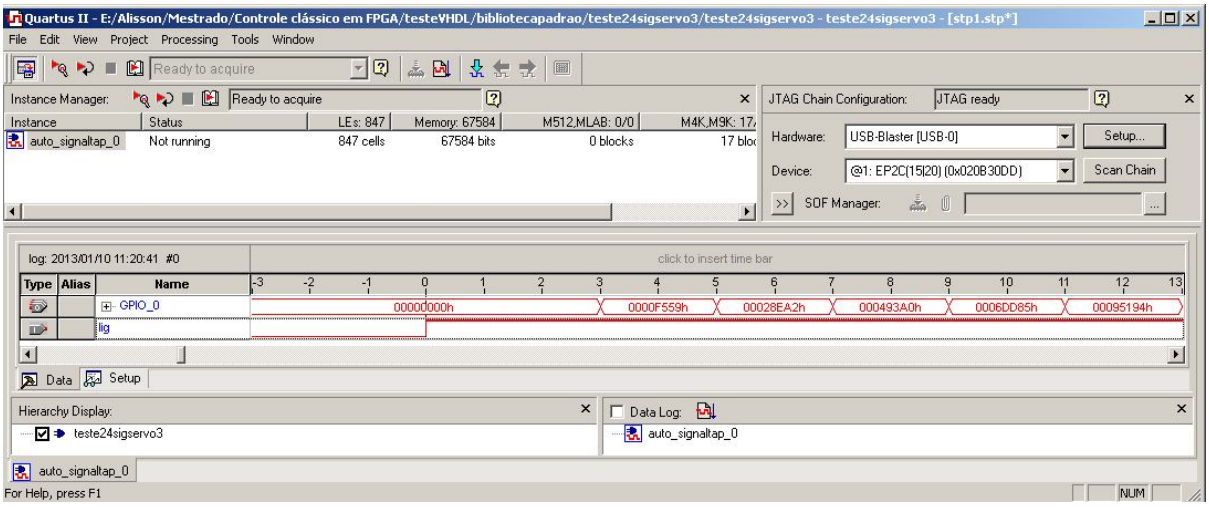


Figura 27 – Tela de coleta de dados do SignalTap II.
Fonte: O Autor.

A questão dos bits de precisão na representação da planta é um item muito relevante. A falta de precisão pode representar o comportamento da planta com um pequeno erro de dinâmica ou mesmo demonstrar um comportamento totalmente contrário ao real. Na figura 29, é exemplificado este comportamento. Na figura 29a é apresentado o comportamento do pêndulo invertido no limiar da representação das casas decimais de precisão. Na figura 29b é usada uma representação com alguns bits a menos. Nesse último caso, observa-se um comportamento totalmente contrário ao comportamento esperado do sistema real.

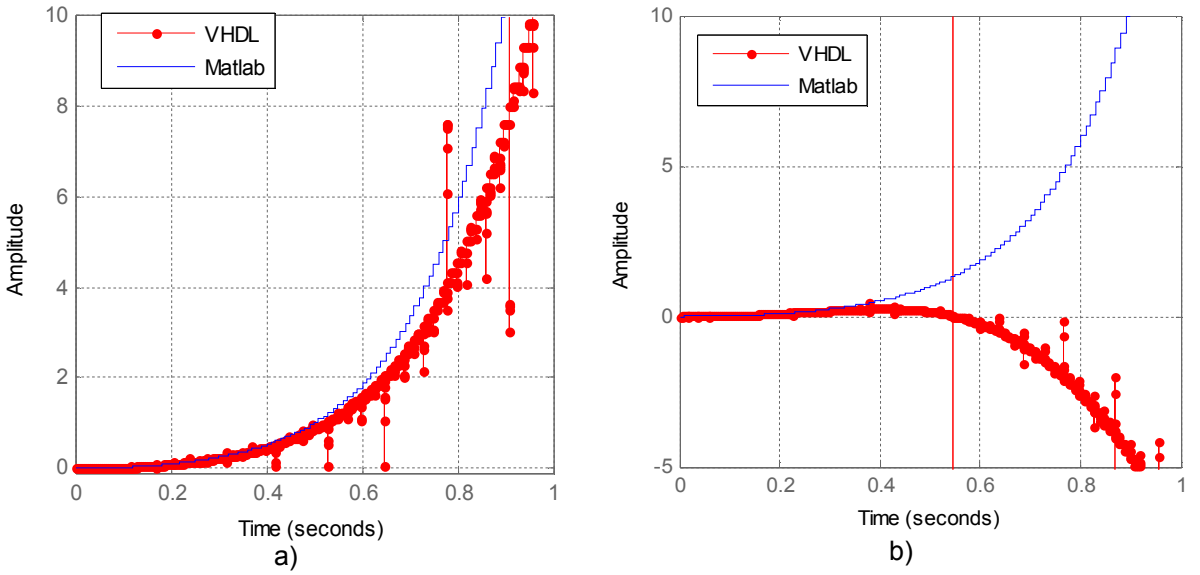


Figura 29 – Apresentação da resposta da simulação com poucos bits de precisão. A) modelo do pêndulo invertido no limiar da representatividade, b) simulação com poucos bits de precisão e sem representatividade.
Fonte: O Autor.

De forma resumida, algumas observações podem ser feita quanto à simulação em *hardware*:

- Todas as plantas escolhidas (forno, servomotor, braço robótico e pêndulo invertido) podem ser adequadamente representadas em *hardware* usando ponto fixo. Isto é particularmente verdadeiro para representação [8:23], definida como padrão IEEE 754.
- Outras representações em ponto fixo podem ser utilizadas. Contudo, deve-se observar criteriosamente os resultados práticos obtidos em função dos elementos lógicos utilizados. Uma representação inadequada pode levar a resultados inaceitáveis.
- Os pontos definidos como *outliers* não são relevantes para o comportamento final do sistema. O sinal de saída do controlador será utilizado para controlar a planta apenas no instante de amostragem, ou seja, onde não ocorrem os *outliers*.

Com base no funcionamento aceitável das plantas quando comparadas com os modelos em Matlab o próximo passo é a aplicação do controlador PID sobre essas plantas. Esses dados são mostrados no próximo subitem.

5.2 EXPERIMENTOS PROPOSTOS PARA A VALIDAÇÃO DO CONTROLADOR EM HARDWARE

O primeiro experimento proposto é a coleta do valor do sinal de controle no primeiro instante de tempo. Este experimento visa observar o comportamento de um controlador em *hardware* construído com diferentes precisões em ponto fixo.

O controlador escolhido apresenta as seguintes características: kp:1.905; ki:0.475; kd:0.175 e dt:0.1. Com base nesses valores uma sequência de testes foi realizada e os dados alcançados estão na tabela 5.

A tabela 5 consiste nos valores fictícios do erro (E) indo de -5 até +5 na primeira coluna. A resposta teórica (exata) é comparada com os cálculos pelo divisor barra (FPGA_/) e pelo divisor do tipo recíproco (FPGA_R) criados por Bishop (2012). As respostas alcançadas com suas respectivas quantidades de bits antes e depois do ponto são indicados e o erro em cada caso. Por exemplo, segundo a representação 05:10 foram usados 5 bits para a parte inteira e 10 bits para a parte fracionária. Os índices de erro e os desvios observados foram pequenos, com tendência a diminuírem caso a precisão das casas decimais aumente.

Tabela 5 - Resultado dos testes estáticos do sinal de controle para o erro na entrada.

		05:05		05:10		10:05		05:05		05:10		10:05	
'E'	Teórico	FPGA_R	Erro	FPGA_R	Erro	FPGA_R	Erro	FPGA_/	Erro	FPGA_/	Erro	FPGA_/	Erro
-5	-18.5125	-19.688	-6.347	-18.530	-0.096	-19.6875	-6.347	-19.6875	-6.347	-18.535	-0.123	-19.688	-6.347
-4	-14.8100	-15.750	-6.347	-14.824	-0.096	-15.75	-6.347	-15.7500	-6.347	-14.828	-0.122	-15.750	-6.347
-3	-11.1075	-11.813	-6.347	-11.118	-0.096	-11.8125	-6.347	-11.8125	-6.347	-11.121	-0.122	-11.813	-6.347
-2	-7.4050	-7.875	-6.347	-7.412	-0.096	-7.875	-6.347	-7.8750	-6.347	-7.414	-0.123	-7.875	-6.347
-1	-3.7025	-3.938	-6.347	-3.706	-0.097	-3.9375	-6.347	-3.9375	-6.347	-3.707	-0.122	-3.938	-6.347
0	0.0000	0.000	0.000	0.000	0.000	0.0000	0.000	0.0000	0.000	0.000	0.000	0.000	0.000
1	3.7025	3.938	-6.347	3.706	-0.096	3.9375	-6.347	3.9375	-6.347	3.707	-0.122	3.938	-6.347
2	7.4050	7.875	-6.347	7.412	-0.096	7.875	-6.347	7.8750	-6.347	7.414	-0.123	7.875	-6.347
3	11.1075	11.813	-6.347	11.118	-0.096	11.8125	-6.347	11.8125	-6.347	11.121	-0.122	11.813	-6.347
4	14.8100	15.750	-6.347	14.824	-0.096	15.75	-6.347	15.7500	-6.347	14.828	-0.122	15.750	-6.347
5	18.5125	19.688	-6.347	18.530	-0.096	19.6875	-6.347	19.6875	-6.347	18.535	-0.122	19.688	-6.347
	Média		-5.770		-0.087		-5.770		-5.770		-0.111		-5.770
	Desvio Padrão		1.914		0.029		1.914		1.914		0.037		1.914

Fonte: O Autor.

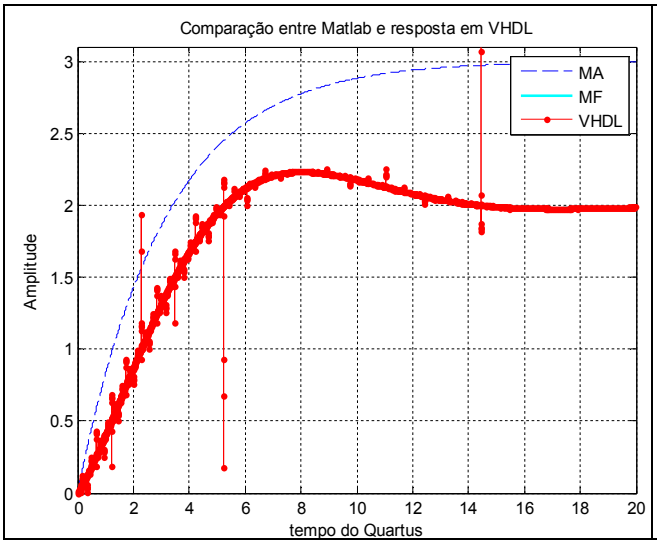
Pela tabela 5, é possível constatar que o erro se manteve constante, uma vez que seu desvio padrão foi pequeno. A tabela 5 apresenta os cálculos com o divisor "/" (FPGA_/) e com o divisor pelo recíproco (FPGA_R) onde a maior diferença ocorreu com o aumento da precisão da parte fracionada. Isso se deve pela perda de precisão do operador divisão "/" conforme comentado no apêndice C.

O segundo experimento trada de um sistema de simulação completo em FPGA onde existe o controlador PID e o modelo da planta, ambos discretizados e com controle sobre a precisão utilizada. O objetivo desse experimento é verificar se o controlador confeccionado em VHDL 2008 consegue agir sobre a planta conforme a simulação em Matlab.

A seguir são apresentados de forma gráfica e numérica os resultados encontrados para as plantas indicadas e os controladores projetados para elas. Nos gráficos são apresentados três itens sendo eles: malha aberta (MA), malha fechada (MF) e VHDL que é o comportamento em malha fechada em FPGA. Além disso, são indicados alguns itens como, por exemplo, índice de erro simulado e do circuito (IAES e IAEC), taxa de amostragem, erro em porcentagem (entre IAES e IAEC) e número de unidades lógicas utilizadas.

Em especial, os elementos lógicos foram contabilizados para cada experimento para gerar uma referência de massa de *hardware* utilizada para cada bit inserido ou retirado da precisão. Isso é importante pois em casos de sistemas complexos onde existem vários controladores em paralelo a quantidade de elementos lógicos (total) utilizadas em uma placa com FPGA pode ser facilmente extrapolada do seu limite impedindo um correto funcionamento.

As figuras 30a, 30b, 30c e 30d são as respostas das plantas de teste, assim como na figura 25 (a-d) só que agora com um controlador PID inserido para melhorar alguma condição de funcionamento, como, por exemplo, zerar o erro em regime permanente.



Planta 1: Forno
Bits: 8:23
Amostragem: 0.01 segundo
IAES: 0.3008
IAEC: 0.3296
Erro(%): 9.55
Elementos lógicos (EL) utilizados: 1371
Valores do PID:
Kp: 0.17400
Ki: 0.18300
KD: 0.00086

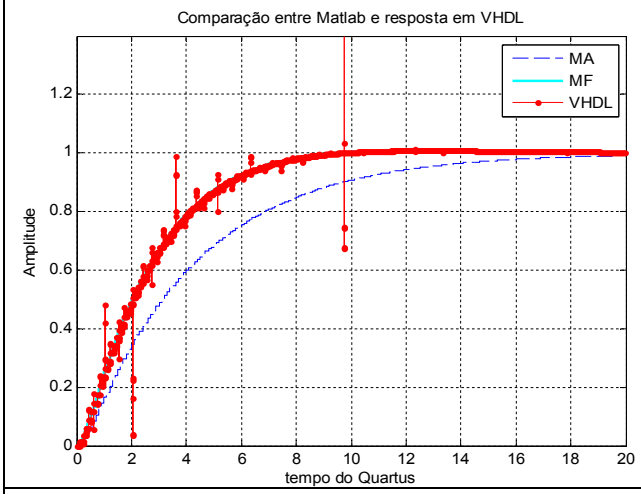
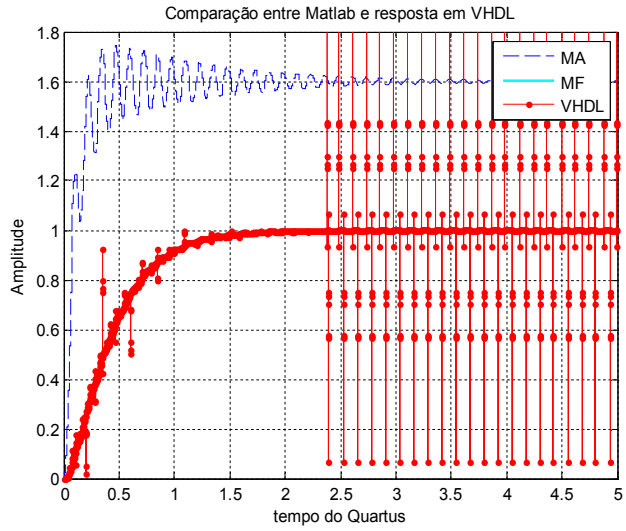
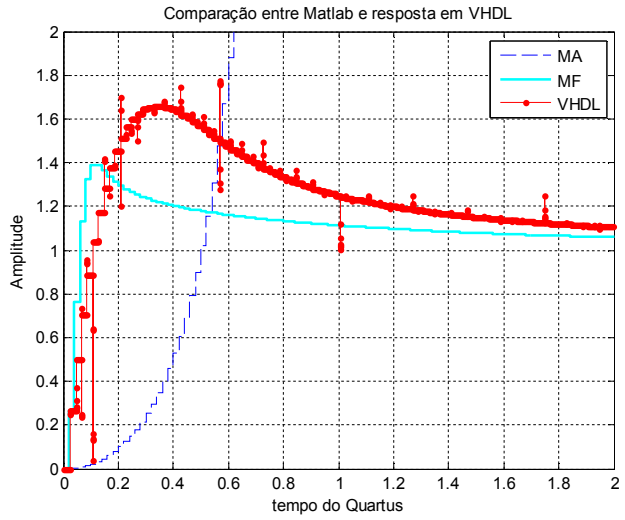
	Planta 2: Servomotor com carga Bits: 8:23 Amostragem: 0.1 segundo IAES: 0.1310 IAEC: 0.1489 Erro(%): 13.66 Elementos lógicos (EL) utilizados: 2010 Valores do PID: Kp: 1.4 Ki: 0.4 KD: 0.06
	Planta 3: Braço de robô Bits: 8:23 Amostragem: 0.01 segundo IAES: 0.0906 IAEC: 0.1296 Erro(%): 43.11 Elementos lógicos (EL) utilizados: 4148 Valores do PID: Kp: 0.014 Ki: 1.380 KD: 0.000 Observação: sistema naturalmente oscilatório
	Planta 4: Pêndulo invertido Bits: 15:52 (valores acima do 8:23) Amostragem: 0.02 IAES: 0.1515 IAEC: 0.3121 Erro(%): 106.04 Elementos lógicos (EL) utilizados: 7683 Valores do PID: Kp: 74.4 Ki: 31.6 KD: 5.4 Observação: sistema naturalmente instável

Figura 30 – Respostas dos circuitos em VHDL 2008 simulando a dinâmica de sistemas físicos com controlador PID - 30a) forno, 30b) servomotor, 30c) braço robótico, 30d) pêndulo invertido Fonte: O Autor.

Tendo em vista o comportamento coerente da simulação dos sistemas em malha aberta e o comportamento com controladores PID implementados em VHDL 2008, conclui-se que o mesmo pode ser aplicado na prática. Contudo, o último

modelo implementado (pêndulo invertido) possui um comportamento diferenciado. Existem alguns itens que podem explicar esse comportamento, como por exemplo a grande variação de amplitude entre as constantes utilizadas nos modelos e os grandes sinais de erro e de controle. O apêndice D dessa dissertação mostra tabelas e gráficos para casos com menor precisão.

6. DISCUSSÕES E CONCLUSÕES

Como início da discussão sobre o trabalho realizado é importante ressaltar a pesquisa inicial com uma linguagem em desenvolvimento e ainda não padronizada: o VHDL 2008. Com base nisso, as ferramentas a serem usadas foram sendo adaptadas às necessidades da área de controle e simulação de sistemas. Para minimizar os erros inerentes destas adaptações algumas precauções foram tomadas:

- Testes realizados com funções de transferência diferentes: optou-se por representações matemáticas de sistemas com dinâmicas diferenciadas, com diferentes ordens de complexidade. Com isto procurou-se evitar eventuais vícios experimentais que poderiam surgir da opção por um único modelo matemático;
- Taxas de amostragem diferenciadas: as taxas de amostragem testadas não foram todas iguais e com isso a região de trabalho engloba aplicações distintas minimizando algum possível erro experimental. Um estudo sobre as taxas de amostragem é indicado no apêndice A;
- Compilação de *hardware* e não apenas simulação: o Quartus II sp1 antes de simular o comportamento do *hardware* precisa sintetizar e criar a estrutura do *hardware* verdadeiro, com isso trazendo mais relevância ao processo de testes. Objetivou-se minimizar os erros gerados por *softwares* que podem alterar o formato dos dados, como por exemplo, não respeitar os bits de precisão nas simulações.
- O controlador PID não é a melhor técnica de controle existente atualmente, mas historicamente, é o mais usado e com isso é possível usar uma estrutura de controle segura e difundida.
- Os pontos definidos como outliers não são relevantes para o comportamento final do sistema. O sinal de saída do controlador será utilizado para controlar a planta apenas no instante de amostragem, ou seja, onde não ocorrem os *outliers*. Caso esses pontos fossem usados nos cálculos a possibilidade da planta não convergir para a estabilidade ou ficar marginalmente estável seria grande, pois esses pontos de alta amplitude induziriam o controlador e gerar valores altos de sinal de controle de forma inconsistente com a dinâmica da planta.

6.1 LIMITAÇÕES E DIFICULDADES ENCONTRADAS

Algumas limitações e dificuldades podem ser listadas:

- A linguagem VHDL 2008 não está finalizada e foram encontradas algumas limitações. Por exemplo, as operações aceitas dentro do comando resize do VHDL 2008;
- O dado do tipo sfixed não aceita valores iniciais, sejam elas do tipo signal, variable ou constantes. Dessa forma o código final, pela criação de linhas exclusivas para indicar os valores usados, demanda linhas adicionais para definição dos valores iniciais;
- O SignalTap II não permite integração com o Simulink de forma sincronizada, isso só é possível com o uso do pacote DSP Builder da Altera, dessa forma, todos os testes com o SignalTap II consomem um tempo adicional;
- O comando resize possui uma limitação de operações que varia com o tamanho do vetor. Recomenda-se dividir operações longas em operações menores para que o resultado final seja correto. É comum o não aparecimento de erro na compilação, mas os resultados de cálculo são inexatos.
- O sistema de simulação do Quartus II apresenta vários itens transitórios que geram pontos fora da curva (*outliers*). Dessa forma, a visualização gráfica indica de forma aparente o surgimento de erros, quando estes não são computados pelo controlador uma vez que acontecem em tempos diferentes dos utilizados na amostragem. Entretanto para fins de cálculo (IAEC) esses *outliers* são computados e dessa forma indicam um erro maior que o real.
- O w*aveform* não apresenta método de controle das amostragens, ele entrega o vetor de simulação conforme parâmetros internos do fabricante. Desse modo torna-se difícil comparar os dados gráficos simulados em Matlab com os gráficos gerados pelo *waveform*. A única opção é usar integralmente o vetor gerado pelo Quartus e comparar com o Matlab.
- Usando o elemento interno SIGNAL é gerado um atraso nos dados, uma vez que o sinal só é usado no final do comando *if* ou *process*. Com isso ele gera um atraso nos cálculos. Para eliminar o problema é necessário usar o comando *shared variable* em todos os dados para que eles sejam computados dentro de um período de amostragem.

- O cálculo e a atualização dos dados devem acontecer em tempos diferentes devido à acomodação dos sinais em transição no *hardware* (massa de hardware). Caso contrário vão existir erros nos cálculos intermediários. A melhor forma de resolver esse problema, encontrada pelo autor, foi com a utilização de uma máquina de estados.

6.2 CONTRIBUIÇÕES DA DISSERTAÇÃO

Descrevem-se algumas contribuições da dissertação:

- O trabalho é inédito, pois não foi encontrada nenhuma referência científica que fale de controle e simulação em *hardware* com a técnica proposta por Bishop (2012).
- A geração de uma estrutura em Matlab com ponto fixo para testes de controle e simulação gerou um *know-how* do autor para futuros trabalhos com desenvolvimento de técnicas de controle discreto com ponto fixo, bem como com representação de modelos discretos de plantas em ponto fixo;
- A arquitetura em VHDL encontrada se mostrou adequada e pode ser usada futuramente em outros modelos matemáticos de plantas ou controladores discretos, uma vez que ela foi criada para ser genérica e de fácil alteração na precisão das variáveis usadas;
- Esse estudo matemático em VHDL para aplicação em controle encontrou erros no atual padrão em desenvolvimento da linguagem que podem contribuir para seu correto funcionamento nas versões definitivas;
- A arquitetura criada permite o estudo do comportamento do sistema mediante a variação da quantidade de bits de precisão, com isso é possível achar uma condição subótima de precisão aonde é possível economizar unidades lógicas sem perder a funcionalidade do controlador. A determinação do ponto ótimo permanece em aberto, sendo um tema de interesse para trabalhos futuros.
- Uma longa lista de testes foi armazenada nos apêndices A, B, C e D para demonstrar todas as precauções tomadas para evitar erros nas conclusões dessa dissertação. Esperasse que essa dissertação seja usada em trabalho futuros na comunidade acadêmica.

6.3 CONCLUSÕES FINAIS

Com base no que foi apresentado até o presente momento concluí-se que o VHDL 2008 com seu sistema de ponto fixo pode ser aplicado na área de controle e simulação de sistemas dinâmicos. Entretanto, entre outros pontos, é importante trabalhar com a técnica de máquinas de estados para minimizar os problemas de atraso (*delay*) de processamento devido à alta massa de *hardware* envolvida no processamento.

Para a implementação de controladores e modelos matemáticos de controle em *hardware* recomendasse o uso prévio de alguma ferramenta de controle e simulação (Matlab, por exemplo) para verificar o comportamento do sistema mediante a alteração da precisão das variáveis e/ou constantes. Erros em regime permanente, erros de projeto como sobre sinal e até mesmo instabilidades podem ocorrer se o sistema implementado em VHDL 2008 não possuir uma precisão mínima para seu funcionamento.

Como já era de se esperar, com o aumento de bits de cálculo no circuito aumenta-se a precisão dos cálculos. Da mesma forma, conforme apêndice D, a quantidade de *hardware* usado (Elementos lógicos) também aumenta. A quantidade de elementos lógicos não foi comentado a fundo nas etapas anteriores uma vez que a linguagem ainda não foi padronizada e, devido a esse motivo, os valores alcançados podem mudar. Todavia, o apêndice D da uma ideia da quantidade de elementos lógicos usados para cada teste realizado. É importante relatar que a quantidade de bits utilizado não pode ser aleatório, pois valores muito altos trazem um pequeno aumento da precisão comparado a uma grande quantidade de *hardware* adicional usado. É indicado sempre analisar qual é a limite da representatividade numérica e deixar o controlador próximo desse valor para minimizar a quantidade de elementos lógicos utilizados.

A tabela 5 onde constam dados de um teste estático somado aos dados da simulação dinâmica demonstram que o controlador PID controla adequadamente a planta quando sua precisão, número de bits, não está no limite da representatividade.

A proposta de operação com ponto fixo de Bishop (2012) é uma contribuição importate na área das FPGAs, pois o código é aberto e funcional. Existem algumas

melhorias a serem feitas como, por exemplo, otimizar o comando *resize* e também corrigir o erro de sinal na divisão que ocorre com o denominador negativo.

6.4 TRABALHOS FUTUROS

Durante o estudo sobre as FPGAs, suas tecnologias e formas de implementar controladores sobre elas, algumas informações relevantes e dúvidas foram encontradas de forma que outros trabalhos, posteriores a este, podem ser realizados a fim de melhorar o que já foi proposto.

Como trabalhos futuros podemos citar:

- Teste com as operações com ponto flutuante e comparar a quantidade de unidas lógicas utilizadas e o ganho de precisão;
- Implementação de outros controladores, como os controladores por modelo interno;
- Desenvolvimento de um *hardware* que possa se comunicar diretamente com o Matlab/Simulink, dessa forma gerando uma plataforma de testes mais completa onde controladores em *hardware* interajam com modelos em *software* no Matlab, inclusive com perturbações e ruídos considerados no modelo;
- Montagem de plantas didáticas de alta velocidade como servomotores para desenvolvimento e testes de controladores paralelos, aproveitando vantagens inerentes das FPGAs;
- Determinar de forma automática, por função, o ponto ótimo da relação entre representação do ponto fixo, tipo da planta e recursos de *hardware*, pois esses itens estão interligados e a alteração de um recai sobre os outros.

REFERÊNCIAS

AL-ASHRAFY, M.; SALEM, A.; ANIS, W. **An Efficient Implementation of Floating Point Multiplier**. IEEE. Electronics, Communications and Photonics Conference (SIECPC), 2011 Saudi International Digital Object Identifier: Publication Year: 2011.

ALECSA B. C., ONEA, A. **An FPGA implementation of the time domain deadbeat algorithm for control applications**. IEEE. NORCHIP. 2009.

ALECSA, B. C. ONEA, A. **An FPGA implementation of a brushless DC motor speed controller**. IEEE16th International Symposium for Design and Technology in Electronic Packaging (SIITME). 2010.

ALECSA, B., ONEA, A. **Design, validation and FPGA implementation of a brushless DC motor speed controller**. 17th IEEE International Conference on Electronics, Circuits, and Systems (ICECS), 2010.

ALONSO, R. M., MINO, K., LUCIO, D. T. **Array Processors Designed with VHDL for Solution of Linear Equation Systems Implemented in a FPGA**. IEEE. Electronics, Robotics and Automotive Mechanics Conference. 2010.

ALTERA. **Quartus II Simulator Tools for Education.** Dipoinível em: < http://www.altera.com/education/univ/software/qsim/unv-qsim.html >. Acesso em: 11 jul. 2012.

ASTRÖM, Karl J.; HAGGLUND, Tore. **Advanced PID control**. Research Triangle Park, NC: ISA, c2006. 460 p.

ASTRÖM, Karl J.; HAGGLUND, Tore. **PID controllers**. 2nd ed. Research Triangle Park, NC: Instrument Society of America, c1995. viii, 343 p.

ASTRÖM, Karl J.; WITTENMARK, Björn. **Computer controlled systems: theory and design.** 3rd. ed. New Jersey: Prentice-Hall, 1997. xv, 557 p. (Prentice-Hall information and system sciences series)

BANATI, N., THIBEAULT, C., GARGOUR, C. S. **An efficient FPGA implementation of a pulse-shaping IIR filter**. IEEE. Electrical and computer Engineering. Canadian conference. 2001. Vol. 1. pag. 353.

BANERJEE, P., BAGCHI, D., HALDAR, M., NAYAK, A. KIM, V., URIBE, R. **Automatic conversion of floating point MATLAB programs into fixed point FPGA based hardware design**. IEEE. 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2003. FCCM 2003.

BISHOP. D., **Fixed point package user's guide.** Disponível em: <http://www.vhdl.org/fphdl/Fixed_ug.pdf> Acesso em: 12 mai. 2012.

BISHOP. D., **Floating point package user's guide**. Disponível em: < http://www.vhdl.org/fphdl/Float_ug.pdf > Acesso em: 12 mai. 2012.

BURDEN, Richard L.; FAIRES, J. Douglas. **Numerical analysis.** 7th ed. Pacific Grove, CA: Thomson Brooks/Cole, c2001. xiii, 841 p.

BUSTAMENTE Eric Zurita, FLORES, Jésus Linares, RAMÍREZ Enrique Guzmán, RAMÍREZ Herbert Sira. **A Comparison Between the GPI and PID Controllers for the Stabilization of a DC–DC "Buck" Converter- A Field Programmable Gate Array Implementation.** IEEE. 2011.

CAVANAGH, Joseph J. F. **Digital computer arithmetic: design and implementation.** New York: McGraw-Hill, 1985. xi, 468 p. (McGraw-Hill computer science series)

CHAN, Y. F., MOALLEM, M., WANG, W. **Efficient implementation of PID control algorithm using FPGA technology**. 43rd IEEE Conference on Decision and Control. CDC 2004. 2004.

CHAN, Y. F., MOALLEM, M., WANG, W. **Efficient implementation of PID control algorithm using FPGA technology**. 43rd IEEE Conference on Decision and Control. CDC 2004. 2004.

CHAPRA, Steven C. **Métodos numéricos para engenharia**. 5. ed. São Paulo: McGraw-Hill, 2008. 809 p.

CHEN, Y., WU, Q. **Design and implementation of PID controller based on FPGA and genetic algorithm**. IEEE. International Coriference on Electronics and Optoelectronics (ICEOE 2011). 2011.

Chiu-Keng Lai, Cih-Ling Chen and Kun-Lin Ho. **A generalized multi-channel PID controller module design using FPGA.** IEEE. 2010

CHOI, Y. CHUNG, W. K. **PID trajectory tracking control for mechanical systms**. Springers. LNCIS. 2004. 298 p.

CUNHA, M. Cristina C. **Métodos numéricos.** 2. ed., rev. e ampl. Campinas: Ed. UNICAMP, c2000. 276 p. (Coleção Livro-texto)

DATTA, A. HO, Ming-Tzu. BATTACHARYYA, S. **Advances in industrial control: Structures and synthesis of PID controller.** Springer. 2000. 235 p.

DATTA, Aniruddha; HO, Ming-Tzu; BHATTACHARYYA, S. P. **Structure and synthesis of PID controllers**. London: Springer, c2000. 233 p.

DORF, Richard C.; BISHOP, Robert H. **Sistemas de controle modernos.** 8.ed. Rio de Janeiro: LTC, 2001. 657 p.

DUFOUR, C. LAPOINTE, V. BELANGER, J. ABOURIDA, S. **Closed-loop control of virtual FPGA-coded permanent magnet synchronous motor drives using a rapidly prototyped controller**. 13th Power Electronics and Motion Control Conference, 2008. EPE-PEMC 2008.

EDA Industry Working Groups. **FAQ**. Disponível em: <http://www.eda.org/fphdl/fpfaq.html>. Acesso em: 5 jan. 2012.

ERCEGOVAC, Milos D., LANG, Tomas. **Digital Arithmetic.** San Francisco: Elseier. 709. 2004.

FALDAR, M, NAYAK, A, CHOUDHARY. A, BANERJEE, P., **A system for synthesizing optimized FPGA hardware from Matlab(R)**. IEEE. 2001.

IDE, T. YOKOYAMA, T. **A study of deadbeat control for three phase PWM inverter using FPGA based hardware controller**. 35 Annual IEEE Power Electronics Specialists Conference. 2004.

IDKHAJINE, L., MONMASSON, E., MAALOUF, A. **Extended Kalman Filter for AC drive Sensorless Speed Controller - FPGA-based solution or DSP-based solution**. IEEE International Symposium on Industrial Electronics. (ISIE 2010), 2010.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE Standard VHDL Language Reference Manual.** Disponível em: < http://www.ece.uic.edu/~dutt/courses/ece368/lect-notes/VHDLref.pdf>. Acesso em: 11 jul. 2012.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **Recent advances in robust control**. New York: IEE, c1990. 501 p.

ITO, S. A., CARRO, L. **A comparison of microcontrollers targeted to FPGA-based embedded applications.** Proc. IEEE 13th Symp. Integrated Circuits and Systems Design, 2000, pp. 397–402.

JASTRZEBSKI, R., POLLANEN, R., PYRHONEN, O. **Analysis of System Architecture of FPGA-based Embedded Controller for Magnetically Suspended Rotor.** IEEE. Internation symposium of System on chip. 2005.

JOHNSON, M. A. MORADI, M. **PID control**: new identification and design methods. Springer. 2005. 539 p.

JÓZMIAK, L.; NEDJAH, N.;FIGUEROA, M. **Modern development methods and tools for embedded reconfigurable systems: A survey.** Elseir.Integration, the VLSI Journal. n. 43. 2010. p. 1-33.

KHAJEHODDIN. S. A., MASOUND, K. G. JAIN, P. BAKHSHAI, A. **A Resonant Controller with High Structural Robustness for Fixed-Point Digital Implementations.** IEEE, 2010. 29 p. NO PRELO.

LAI, C., CHEN, C., HO, K. **A generalized multi-channel PID controller module design using FPGA**. IEEE. International Symposium on Computer Communication Control and Automation (3CA), 2010.

LÁZARO, J., ARIAS, J. MARTÍN, J. L. ALEGRIA, I. M. ANDREU, J., JIMENEZ, J., **An implementation of a general regression network on FPGA with direct Matlab link.** IEEE. INternational conference on industrial techonolgy (ICIT). 2004.

LEE, W. K., JUNG, S. **FPGA Design for Controlling Humanoid Robot Arms by Exoskeleton Motion Capture System**. International Conference on Robotics and Biomimetics. ROBIO '06. 2006.

LEONG, M. P., YEUNG, M. Y. YEUNG, C. K. FU, C. W. HENG, P. A. LEONG, P. H. W. **Automatic floating to fixed point translation and its application to post-rendering 3D warping**. IEEE. Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 1999. FCCM '99.

MAXFIELD, Clive. **The design warrior's guide to FPGAs:** devices, tools, and flows. Boston: Newnes/Elsevier, c2004. xvi, 542 p.

MAXON. **Epos2 Apllication Notes Collection**. 218 p. 2011.

MONMASSON, E., CIRTEA, M. N. **FPGA Design Methodology for Industrial Control Systems - A Review.** IEEE Transactions on Industrial Electronics, 2007.

MOREIRA, J. C. FARRELL, P. G. **Essentials of error-control coding.** Chichester, ENG: J. Wiley, c2006. 361 p.

O'DWYER, Aidan. **Handbook of PI and PID controller tuning rules**. 2nd ed. London: Imperial College Press, 2006. xvii, 545 p.

OGATA, Katsuhiko. **Engenharia de controle moderno.** 3. ed. Rio de Janeiro: Prentice-Hall, c1998. xiv, 813 p.

OLIVEIRA, A. A., LIMA, C. R. E., LOPES, H. S. L. **Projeto de controladores PID em sistemas multivariáveis usando computação evolucionária**. 10th Brazilian congress on computational intelligence (CBIC'2011). 2011.

PEDRONI, Volnei A. **Circuit design with VHDL.** 2nd Ed. Cambridge, Mass.: MIT Press, 2009. 608 p.

PEDRONI, Volnei A. **Circuit design with VHDL.** Cambridge, Mass.: MIT Press, c2004. 363 p.

PEDRONI, Volnei A. **Eletrônica digital moderna e VHDL.** Ed. Campus, 2010. 619 p.

PHILLIPS, Charles L.; NAGLE, H. Troy. **Digital control system analysis and design.** 2nd ed. Englewood: Prentice-Hall, c1995. 685 p.

PHILLIPS, Charles L.; NAGLE, H. Troy. **Digital control system analysis and design.** 2nd ed. Englewood: Prentice-Hall, c1995. 685 p.

QU, L., HUANG, Y., LING, L. **Design and Implementation of Intelligent PID Controller Based on FPGA**. Fourth International Conference on Natural Computation. ICNC '08. 2008.

QUEVEDO, J.; ESCOBET, T. INTERNATIONAL FEDERATION OF AUTOMATIC CONTROL; IFAC WORKSHOP ON DIGITAL CONTROL : PAST, Present, and Future of PID Control : (2000 Terrassa,

Spain) (Ed.). **Digital control: past, present, and future of PID control** (PID '00) : a proceedings volume from the IFAC Workshop, Terrassa, Spain, 5-7 April 2000. New York: Pergamon, 2000. 608 p.

ROSÁRIO, João Maurício. **Princípios de mecatrônica.** São Paulo: Prentice-Hall, 2005. x, 356 p.

SANTINA, M. S., STUBBERUD, A. R., HOSTETTER, G. H. Efeitos da quantização. **In:** The Control Handbook 1996 p. 301.

SILVA, G. J. DATTA, A. BHATTACHARYYA, S. P. **PID controllers for time-delay systems**. Control Engineering. 2005. 329 p.

SREENIVASAPPA, B. V., UDAYKUMAR, R. Y. **Design and Implementation of FPGA Based Low power digital controllers**. Fourth International Conference on Industrial and Information Systems, ICIIS 2009. 2009.

TAKAMATSU, S., SHIMADA, E., YOKOYAMA, T. **Digital Control Method for Single Phase Utility Interactive Inverter Using Deadbeat Control with FPGA Based Hardware Controller**. 12th International Power Electronics and Motion Control Conference. EPE-PEMC 2006. 2006.

Tang, Xiaoming. Zhang, Tao. Wang, Zhenjie. Yuan , Wen L. **A novel data format conversion method based on FPGA.** IEEE. 2011.

TAO, Y., LIN, H., HU, Y., ZHANG, X. WANG, Z. **Efficient implementation of CNC Position Controller using FPGA**. 6th IEEE International Conference on Industrial Informatics. INDIN 2008. 2008.

TEIXEIRA, P. R. F. **Sintonia de Malhas de Controle (PID).** Curso de Instrumentação Industrial. Módulo VI. ISA. Curitiba. 2012. 65 p.

THE CONTROL handbook. Boca Raton, FL.: CRC Press, 1996. 1548 p.

THE MATHWORK. **EDA Simulator Link**. Disponível em: <http://www.mathworks.com/products/eda-simulator/>. Acesso em: 24 nov. 2011.

THE MATHWORK. **HDL Cosimulation** . Disponível em: <http://www.mathworks.com/products/hdl-verifier/description2.html>. Acesso em: 24 nov. 2011.

THE MATHWORK. **Simulink HDL Coder**. Disponível em: <http://www.mathworks.com/products/slhdlcoder/>. Acesso em: 24 nov. 2011.

THULASIRAMAN, N. K., MOHAMED, H. A. F., CHENG, Y. S. **A Reconfigurable Wireless Stepper Motor Controller Based On FPGA Implementation.** Symposium on Industrial Electronics and Applications. (ISIEA 2010) 2010.

URRIZA, I., BARRAGAN, L. A., ARTIGAS, J. I. NAVARRO, D. LUCIA, O. **FPGA Implementation of a Digital Controller for a dc-dc Converter Using Floating Point Arithmetic**. IEEE. 35th Annual Conference of Industrial Electronics, 2009. IECON '09. 2009.

VHDL ORG. **FPGA FAQ**. Disponível em: <http://www.eda.org/fphdl/fpfaq.html>. Acesso em: 05 mai. 2012.

VILANOVA, Ramon; VISIOLI, Antonio. **PID Control in the Third Millennium:** Lessons Learned and New Approaches Advances in Industrial Control. Springer. 2012. 599 p.

VISIOLI, A. **Pratical PID Control. Advanced in Industrial Control**. Springer. 2006.

WANG, Qing-guo. YE, Z. CAI, Wen-Jian. HANG, Chang-Chieh. **PID control of multivariable processes.** LNCIS. Springer. 2008. 373 p.

WILKIE, J., JOHNSON, M. A., KATEBI, R. **Control Engineering: An introductory course**. New Yourk. Ed. Polgrave Macmillan. 2002. p. 750.

WU, Dan., CHEN, Ken. **Active disturbance rejection approach with application to precision motion control**. IEEE. 30th Chinese Control Conference. 2011.

YAU, H., LIN, M., CHAN, Y., YUAN, K. **Design and implementation of real-time NURBS interpolator using a FPGA-based motion controller**. IEEE. International Conference on Mechatronics, ICM '05. 2005.

YU, Cheng-Ching. **Autotuning of PID controllers: relay feedback approach**. London: Springer, c1998. 226 p. (Advances in industrial control)

ZHANG, Da, LI, HUI. **A Stochastic-Based FPGA Controller for an Induction Motor Drive With Integrated Neural Network Algorithms**. IEEE Transactions on industrial electronics, vol. 55, no. 2, February. 2008. p. 551.

TOCCI, Ronald J.; WIDMER, Neal S.; MOSS, Gregory L. **Sistemas digitais:** princípios e aplicações. 10. ed. São Paulo: Pearson Prentice Hall, c2007. xxii, 804 p.

SILVA FILHO, O. S. Estratégias sequenciais subótimas para planejamento agregado da produção sob incertezas. **In:** Revista Gestão e Produção. v.7, n.3, p.247-268, dez. 2000.

APÊNDICE A – Estudo dos métodos de discretização para seleção do usado na dissertação

Estudo dos métodos de discretização

Devido as técnica de discretização de sistemas de controle possuírem suas características próprias, e existirem indicações sobre as técnicas usadas, foi levantada a seguinte pergunta, qual é a melhor? Como base nisso foi feito um estudo para achar qual é a melhor técnica de discretização e para qual condição. Esse estudo contempla apenas funções de transferência usadas em controle e não a técnica de espaço de estados, pois isso iria inserir variáveis que iriam no mínimo dobrar a quantidade de experiências e análises feitas. Como o estudo em FPGA se deu com equações a diferenças e não espaço de estados é aceitável restringir o estudo a essa técnica.

Com base nisso é apresentado um estudo envolvendo 3 modelos matemáticos que representam 3 processos/máquinas que foram analisados com 4 métodos e cada uma com 4 taxas de amostragem. Dessa forma foram realizados 48 testes para então escolher o melhor método de discretização para ser implementado em sistemas de controle discreto.

Os métodos de discretização são:

- ZOH: O segurador de ordem zero (*Zero Order Hold*) é o mais básico e simples método e discretização. Esse simula o tempo que os processadores ficam sem "visualizar" as entradas de forma que é retida a última amostra recebida até que uma nova seja recebida;
- FOH: o segurador de primeira ordem (*First Order Hold*), também chamado de interpolação linear, se difere do ZOH por gerar uma representação de primeira ordem, como uma rampa, para cada amostragem, com isso existe uma projeção do comportamento entre os períodos de amostragem.
- Tustin: O método de Tustin também conhecido como método trapezoidal ou aproximação bilinear trabalha com as diferenças e é comumente aplicado a resposta ao impulso.
- Matched: o método de zeros e polos de Matched é muito simples. Ele simplesmente faz a seguinte conversão entre digital (z) e contínuo (s) dos pólos e zeros: $Z = e^{sT}$ onde T é o período de amostragem.

A análise ocorrerá de duas formas, na primeira com a visualização gráfica das respostas aos modelos gerados com uma entrada do tipo degrau unitário (constante e igual a 1) para visualizar o período transitório e em regime. A segunda análise será feita usando dois índices conhecidos da área de controle que são o IAE e ISE (integral do erro absoluto e integral do erro absoluto ao quadrado, respectivamente) que não estão sendo usados para ver a qualidade da resposta, mas sim para mostras as variações das respostas para cada um dos métodos apresentando, e tendo como elemento de comparação final um simples número. Quanto maior for à diferença entre esses números mais diferença existe entre as curvas do gráfico.

1 MODELO SOBRE ANÁLISE 01 – MOTOR DE CORRENTE CONTÍNUA

O primeiro sistema estudado é do tipo eletromecânico, sistema de segunda ordem sem sobre sinal ou tempo morto. O sistema é um motor de corrente contínua (CC) da empresa Maxon, tal motor foi usado devido à indicação do modelo matemático pelo próprio fabricante, o modelo se encontra em seu d*atasheet* (MAXON. 2011).

Esse motor foi selecionado pela possibilidade de sua atuação como servomotor, que são sistemas de alta complexidade necessitando de controladores rápidos para o controle da posição, velocidade, aceleração e em alguns casos o “jerk”, que é o “tranco” do motor (estudado em máquinas ferramenta como CNC).

A função de transferência contínua que relaciona a velocidade do eixo pela tensão de entrada é apresentada na função 1.

$$H(s) = \frac{9{,}168 * 10^6}{1622s^2 + 6359s + 1471} \quad (1)$$

A figura 1 mostra o comportamento do motor para sua tensão nominal com uma carga de inércia no valor de 5000 gcm^2. Os gráficos apresentados mostram o comportamento simulado para 5 casos: modelo contínuo e mais quatro amostragens, que são as indicadas acima (Tustin, Matched, ZOH e FOH).

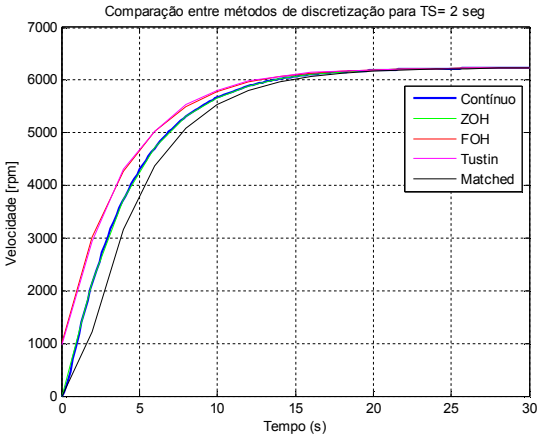


Figura 1 - Gráfico do comportamento do sistema com amostragens de 2 segundos e modelo contínuo.

A figura 1 mostra o comportamento do motor amostrado com 2 segundos. A figura 1 demonstra de forma fácil os erros no período transitório com essa baixa taxa de amostragem. A tabela 1 apresenta os índices com valores diferentes, o que demonstra numericamente os comportamentos diferenciados entre as curvas que representam os modelos discretizados.

Tabela 1 - Tabela comparativa com a área do erro durante o período transitório do modelo.

Método	IAE	ISE	Ordem *[num;den]
Contínuo	1,47E+05	4,96E+08	[0;2]
ZOH	3,33E+04	1,31E+08	[1;2]
FOH	2,68E+04	8,72E+07	[2;2]
Tustin	2,68E+04	8,88E+07	[2;2]
Matched	3,81E+04	1,58E+08	[1;2]

*num e den são as abreviações de numerador e denominador

A figura 2 trata da amostragem com 1 segundo de intervalo.

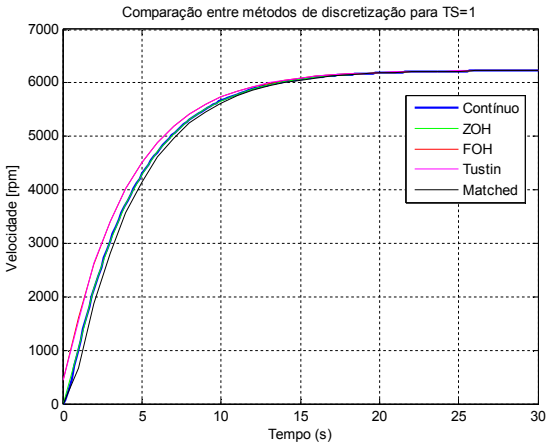


Figura 2 - Gráfico do comportamento do sistema com amostragens de 1 segundo e modelo contínuo.

A tabela 2 apresenta os resultados para a amostragem de 1 segundo.

Tabela 2 - Tabela comparativa com a área do erro durante o período transitório do modelo

MÉTODO	IAE	ISE	ORDEM *[num;den]
Contínuo	1.4700e+005	4.9575e+008	[0;2]
ZOH	2.9950e+004	1.0848e+008	[1;2]
FOH	2.6812e+004	8.8381e+007	[2;2]
Tustin	2.6813e+004	8.8759e+007	[2;2]
Matched	3.1636e+004	1.1864e+008	[1;2]

*num e den são as abreviações de numerador e denominador.

A mesma sequência de testes foi aplicada com outro valor de amostragem e comparada, agora com uma amostragem de 0.1 segundo (100ms). A figura 3 apresenta a resposta transitória e em regime.

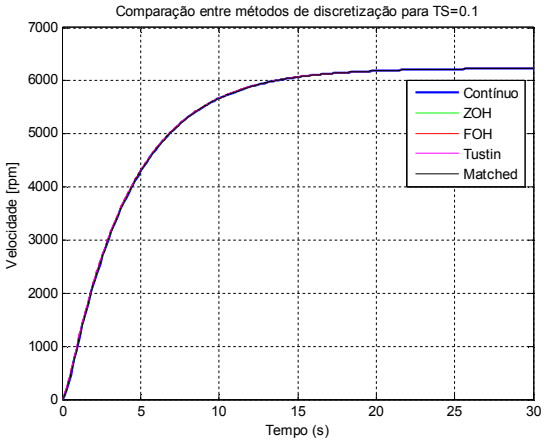


Figura 3 - Gráfico do comportamento do sistema com amostragens de 0.1 segundo e modelo contínuo.

Novamente os índices IAE e ISE são apresentados, para essa nova amostragem esses dados são apresentados na tabela 3.

Tabela 3 - Tabela comparativa com a área do erro durante o período transitório do modelo

Método	IAE	ISE	Ordem *[num;den]
Contínuo	1,4700e+005	4,9575e+008	[0;2]
ZOH	2,7123e+004	9,0700e+007	[1;2]
FOH	2,6811e+004	8,8755e+007	[2;2]
Tustin	2,6811e+004	8,8759e+007	[2;2]
Matched	2,7143e+004	9,0826e+007	[1;2]

*num e den são as abreviações de numerador e denominador

Agora com tempo entre amostragens de 0.001 segundo (1ms)

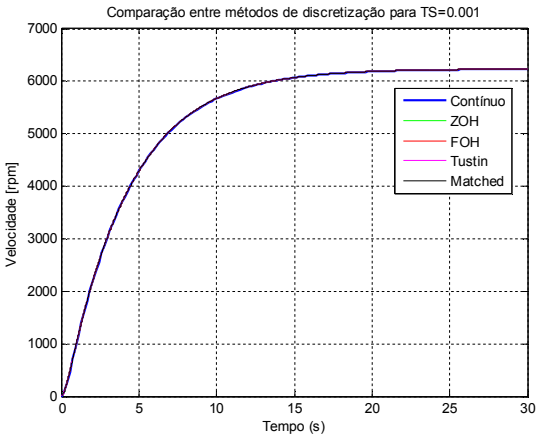


Figura 4 - Gráfico do comportamento do sistema com amostragens de 0.001 segundo e modelo contínuo.

A tabela quatro apresenta os resultados para a amostragem de 0.001 segundo.

Tabela 4 - Tabela comparativa com a área do erro durante o período transitório do modelo.

Método	IAE	ISE	Ordem *[num;den]
Contínuo	1.4700e+005	4.9575e+008	[0;2]
ZOH	2.6814e+004	8.8779e+007	[1;2]
FOH	2.6811e+004	8.8759e+007	[2;2]
Tustin	2.6811e+004	8.8759e+007	[2;2]
Matched	2.6814e+004	8.8779e+007	[1;2]

*num e den são as abreviações de numerador e denominador

Nesse primeiro modelo, que é um modelo de segunda ordem que representa um sistema eletromecânico, é possível concluir que quanto menor o período de amostragem menor é o erro. A diferença observada entre as técnicas de amostragem é que quando comparadas, o ZOH e o Matched formam modelos matemáticos discretos mais simples, com apenas um zero e dois pólos enquanto que o FOH e Tustin formam sistemas com 2 zeros e 2 pólos, ou seja, mais complexos.

Segue-se agora com o segundo modelo em estudo.

2 MODELO SOBRE ANÁLISE 02 – FORNO EM UMA INDÚSTRIA QUÍMICA

O segundo modelo sobre análise é um forno da empresa química WJK apresentado por Wilkie *et al*. (p. 587) O modelo é apresentado na equação 2. A escolha por esse modelo se deu pela exemplificação de sistemas químicos que envolvem aquecimento e tempo motor.

$$H(s) = \frac{3}{3.1s + 1} * e^{-0.4s} \quad (2)$$

A figura 5 mostra o comportamento do sistema para as 4 formas de amostragem com um período de 2 segundos. O gráfico mostra que, principalmente no período transitório, os comportamento são diferenciados, isso se deve a constante de tempo do sistema e de seu tempo morto que estão abaixo da taxa de amostragem. Em termos leigos a planta está se “movendo rápido” e o “observador” não consegue acompanhar suas mudanças.

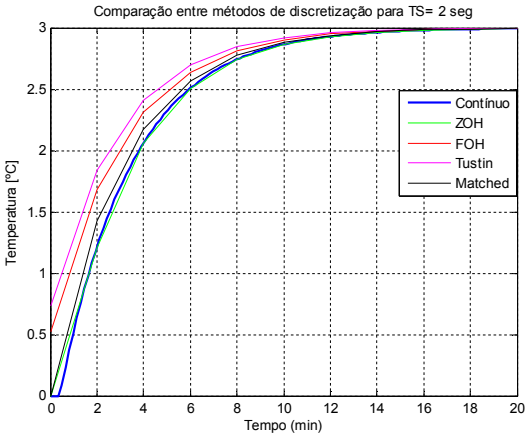


Figura 5 - Gráfico do comportamento do sistema com amostragens de 2 segundos e modelo contínuo.

A tabela 5 mostra os resultados da figura 5 de forma numérica.

Tabela 5 - Tabela comparativa com a área do erro durante o período transitório do modelo 2.

Método	IAE	ISE	Ordem *[num;den]
Contínuo	65,9686	112,3938	[0;1]
ZOH	13,5203	26,8458	[1;1]
FOH	10,4912	17,0524	[2;2]
Tustin	9,2941	13,9500	[1;1]
Matched	12,6099	24,8338	[0;1]

*num e den são as abreviações de numerador e denominador

A figura 6 mostra a mesma comparação do caso do forno para um tempo de amostragem de 1 segundo.

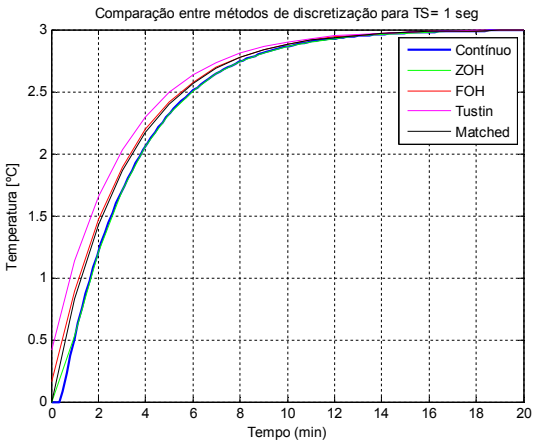


Figura 6 - Gráfico do comportamento do sistema com amostragens de 1 segundo e modelo contínuo.

A tabela 6 mostra os resultados da figura 6 de forma numérica.

Tabela 6 - Tabela comparativa com a área do erro durante o período transitório do modelo.

Método	IAE	ISE	Ordem *[num;den]
Contínuo	65,9686	112,3938	[0;1]
ZOH	11,9517	21,8543	[1;1]
FOH	10,4879	17,4369	[2;2]
Tustin	9,2900	13,9500	[1;1]
Matched	10,8681	18,9305	[0;1]

*num e den são as abreviações de numerador e denominador

A figura 7 mostra o comportamento do forno para uma amostragem de 0.1 segundo.

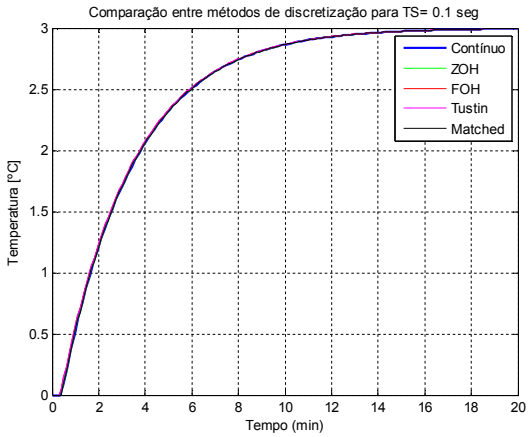


Figura 7 - Gráfico do comportamento do sistema com amostragens de 0.1 segundo e modelo contínuo.

A tabela 7 mostra os resultados da figura 7 de forma numérica.

Tabela 7 - Tabela comparativa com a área do erro durante o período transitório do modelo.

Método	IAE	ISE	Ordem *[num;den]
Contínuo	65,9686	112,3938	[0;1]
ZOH	10,6344	18,0048	[1;1]
FOH	10,4838	17,5487	[2;2]
Tustin	10,4838	17,5500	[1;1]
Matched	10,6344	18,0048	[0;1]

*num e den são as abreviações de numerador e denominador

A figura 8 mostra o comportamento do forno para uma amostragem de 0.001 segundo.

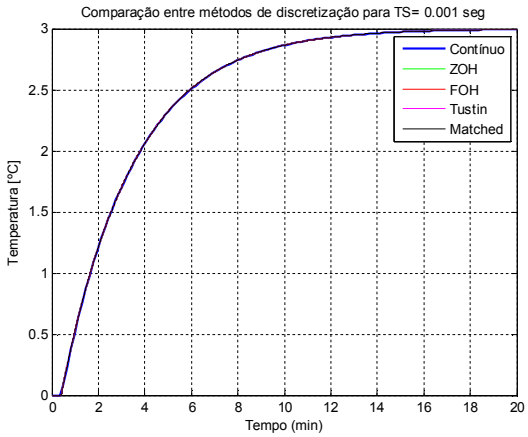


Figura 8 - Gráfico do comportamento do sistema com amostragens de 0.001 segundo e modelo contínuo.

A tabela 8 mostra os resultados da figura 8 de forma numérica.

Tabela 8 - Tabela comparativa com a área do erro durante o período transitório do modelo.

Método	IAE	ISE	Ordem *[num;den]
Contínuo	65,9686	112,3938	[0;1]
ZOH	10,4848	17,5545	[1;1]
FOH	10,4833	17,5500	[2;2]
Tustin	10,4833	17,5500	[1;1]
Matched	10,4848	17,5545	[0;1]

*num e den são as abreviações de numerador e denominador

Essa segunda análise demonstrou que novamente à medida que diminuímos o período de amostragem as técnicas de amostragem se assemelham em comportamento, entretanto as técnicas de Matched e ZOH continuam devolvendo os modelos matemáticos (funções de transferência) mais simples.

Segue-se agora com a última análise.

3 MODELO SOBRE ANÁLISE 03 - PRESSÃO

O terceiro modelo sobre análise é um sistema de refrigeração usado na geração de energia da indústria Clyde Power (Clyde Power glasgow , Scotland) apresentado por Wilkie e*t al*. (p. 610). O modelo é apresentado na equação 3. Esse modelo foi escolhido para estudo devido a sua rápida variação, característica natural das máquinas que trabalham com pressão, que implica em uma boa aplicação para sistema de controle de alta velocidade e por se tratar de um caso real.

$$H(s) = \frac{20(s^2 + 2s + 2)}{(s+3)^3(s^2 + s + 4.25)} \quad (3)$$

O sistema em estudo possui um numerador (zero) de segunda ordem e um denominador (pólo) de quinta ordem, isso demonstra que o sistema é complexo, possui um comportamento complexo. Graficamente é possível visualizar que seu tempo de subida em malha aberta é de aproximadamente 1 segundo, ou seja, é um sistema muito rápido o que demonstra uma boa aplicação para controladores rápido criados com DSPs e FPGAs

A figura 9 apresenta o comportamento da planta de pressão analisada.

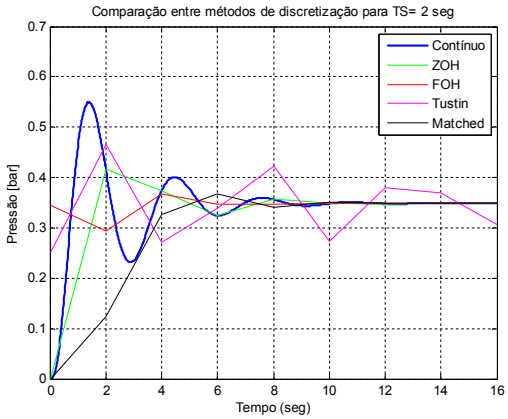


Figura 9 - Gráfico do comportamento do sistema com amostragens de 2 segundos e modelo contínuo.

A tabela 9 mostra os resultados da figura 9 de forma numérica.

Tabela 9 - Tabela comparativa com a área do erro durante o período transitório do modelo.

Método	IAE	ISE	Ordem *[num;den]
Contínuo	33,0163	5,3056	[2;5]
ZOH	0,9572	0,2563	[4;5]
FOH	0,1800	0,0069	[5;5]
Tustin	1,0927	0,0878	[5;5]
Matched	1,2642	0,3495	[4;5]

*num e den são as abreviações de numerador e denominador

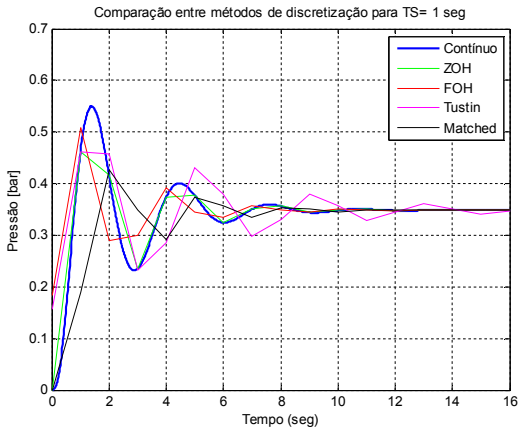


Figura 10 - Gráfico do comportamento do sistema com amostragens de 1 segundo e modelo contínuo.

A tabela 10 mostra os resultados da figura 10 de forma numérica.

Tabela 10 - Tabela comparativa com a área do erro durante o período transitório do modelo.

Método	IAE	ISE	Ordem *[num;den]
Contínuo	33,0163	5,3056	[2;5]
ZOH	0,7418	0,1545	[4;5]

FOH	0,5212	0,0615	[5;5]
Tustin	0,8635	0,0917	[5;5]
Matched	0,7048	0,1582	[4;5]

*num e den são as abreviações de numerador e denominador

A figura 11 apresenta o comportamento do sistema para todas as amostragens da planta de pressão, com o período de amostragem de 0.1 segundo os comportamentos já se assemelham.

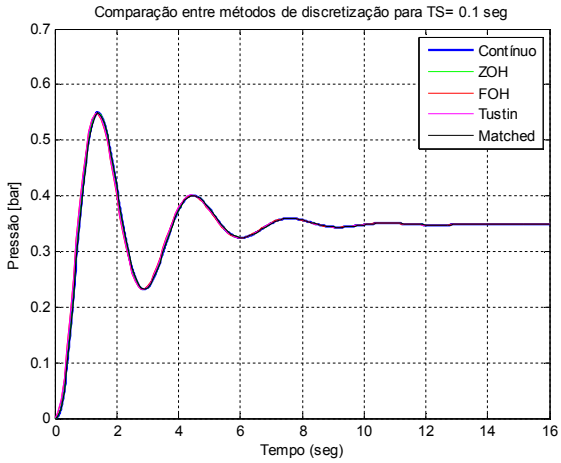


Figura 11 - Gráfico do comportamento do sistema com amostragens de 0.1 segundo e modelo contínuo.

A tabela 11 mostra os resultados da figura 11 de forma numérica.

Tabela 11 - Tabela comparativa com a área do erro durante o período transitório do modelo.

Método	IAE	ISE	Ordem *[num;den]
Contínuo	33,0163	5,3056	[2;5]
ZOH	0,5905	0,0978	[4;5]
FOH	0,5742	0,0914	[5;5]
Tustin	0,5781	0,0917	[5;5]
Matched	0,5909	0,0980	[4;5]

*num e den são as abreviações de numerador e denominador

A figura 12 apresenta o comportamento simulado do sistema para uma taxa de amostragem de 0.001 segundo.

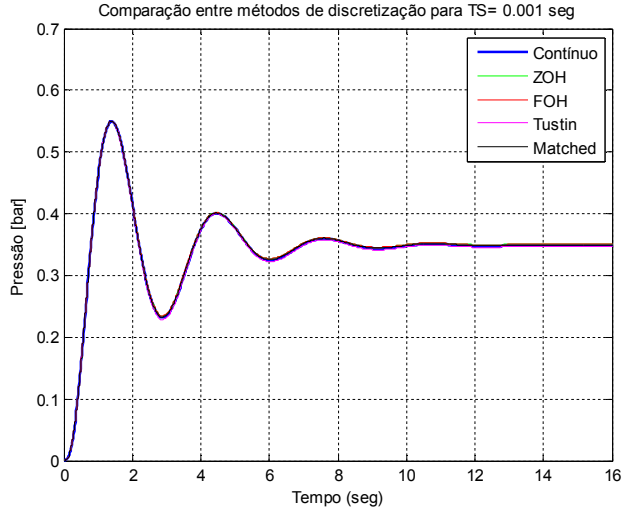


Figura 12 - Gráfico do comportamento do sistema com amostragens de 0.001 segundo e modelo contínuo.

A tabela 12 mostra os resultados da figura 12 de forma numérica.

Tabela 12 - Tabela comparativa com a área do erro durante o período transitório do modelo.

Método	IAE	ISE	Ordem *[num;den]
Contínuo	33,0163	5,3056	[2;5]
ZOH	0,5663	0,0914	[4;5]
FOH	0,5660	0,0912	[5;5]
Tustin	0,5912	0,0924	[5;5]
Matched	0,5691	0,0916	[4;5]

*num e den são as abreviações de numerador e denominador

Essa terceira análise demonstrou que novamente à medida que diminuímos o período de amostragem as técnicas de amostragem se assemelham em comportamento, entretanto as técnicas de Matched e ZOH continuam devolvendo os modelos matemáticos (funções de transferência) mais simples.

Segue-se agora com a conclusão final

4 CONCLUSÃO

Com base nesses 3 modelos com características próprias de velocidade, tempo e ordem ao longo desses 48 testes conclui-se que o ZOH é o método de discretização que apresenta a melhor relação custo benefício, ou seja, ele é simples, tem boa precisão e altera menos o modelo matemático quando comparado ao Tustin ou FOH. Devido a isso nos testes da dissertação será usado o método ZOH de discretização para implementação em FPGA.

APÊNDICE B – conversões numéricas em Matlab com ponto fixo

Conversões numérica em Matlab com ponto fixo

O Matlab trabalha com as seguintes bases numéricas: Double; Single; Int8; Uint8; Int16; Uint16; Int32; Uint32; Fixdt(1,16,0); Fixdt(1,16,2^0,0). Dentre estas, a dissertação foca nos números com ponto fixo por utilizarem menos *hardware* em sua implementação.

O comando "fi" do Matlab permite a transformação do formato Double, que é o padrão do Matlab, para números com ponto fixo. A sintaxe do comando é fi (numero para conversão, sinal, inteiro, fração). Para indicar qual é a forma de tratar os dados recebidos o construtor "fi" é usado.

As possibilidades de configuração são: CastBeforeSum; MaxProductWordLength; MaxSumWordLength; OverflowMode; ProductBias; ProductFixedExponent; ProductFractionLength; ProductMode; ProductSlope; ProductSlopeAdjustmentFactor; ProductWordLength; RoundMode; SumBias; SumFixedExponent; SumFractionLength; SumMode; SumSlope; SumSlopeAdjustmentFactor; SumWordLength.

Nesse trabalho será comentado apenas os que foram utilizados. São eles:

- MaxProductWordLength: tamanho máximo da resposta encontrada após o produto de dois números;
- MaxSumWordLength: tamanho máximo da resposta encontrada após a operação de soma;
- OverflowMode: nesse caso a descrição da aceitação ou não o overflow sobre os dados;
    - Saturate: essa configuração impõem limites máximos e mínimos para não permitir os overflows.
    - Wrap: essa opção permite a ocorrência de overflow e caso ele aconteça será pela técnica do complemento de 2.
- ProductMode: define como será o cálculo do produto, para a operação de dois números de ponto fixo [Wa Fa] e [Wb Fb], suas opções são:
    - FullPrecision: a parte inteira e a parte fracionada são somadas separadamente e um erro ocorre se esse valor for superior ao valor estipulado em MaxProductWordLength.

$$Wp = Wa + Wb$$

$$Fp = Fa + Fb$$

    - KeepLSB: o foco é no bit menos significativo e os bits que sobrarem são usados para compor a parte inteira. Essa formatação pode gerar overflow.
    - KeepMSB: nesse caso o foco é na parte inteira e os bits que sobrarem são usados para compor a fração. Nesse modelo não existe overflow, mas existe perda de precisão.
    - SpecifyPrecision: nesse caso a precisão é especificada com base no ProductWordLength para a parte inteira e no ProductFractionLength para a fração.
- ProductWordLength: esse parâmetro indica o valor máximo aceito no tamanho da resposta encontrada nas operações de multiplicação;
- RoundMode: esse parâmetro descreve a forma de arredondamento aplicada caso ela seja existente;
    - Convergent: Arredonda o valor para o valor inteiro mais próximo.
    - Fix: arredonda para o valor mais próximo de zero (trunca).
    - Nearest: Arredonda para o valor mais próximo positivo (default do Matlab).

- Floor: é equivalente ao “truncamento” do complemento de 2.
- Round: arredonda para o valor mais próximo sendo ele positivo ou negativo.

- SumMode: a forma como será realizada a soma, ele possui as mesmas configurações que o ProdutcMode.
- SumWordLength: tamanho máximo em bits (o default é 32).

Por padrão a ferramenta “fi” do Matlab vem com a seguinte especificação:

RoundMode: nearest
OverflowMode: saturate
ProductMode: FullPrecision
MaxProductWordLength: 128
SumMode: FullPrecision
MaxSumWordLength: 128

A tabela 1 apresenta um exemplo dessas aproximações com a constante PI (π) que é uma constante no Matlab equivalente a 3.14159265358979.

Tabela 1 – exemplo de aproximação pelo comando de número com ponto fixo do Matlab.

PI	Representação	Erro (%)
3.14159265358979	Double	0.0000
3.00000000000000	fi(pi,1,16,0)	4.5070
3.00000000000000	fi(pi,1,16,1)	4.5070
3.25000000000000	fi(pi,1,16,2)	-3.4507
3.12500000000000	fi(pi,1,16,3)	0.5282
3.12500000000000	fi(pi,1,16,4)	0.5282
3.15625000000000	fi(pi,1,16,5)	-0.4666
3.14062500000000	fi(pi,1,16,6)	0.0308
3.14062500000000	fi(pi,1,16,7)	0.0308
3.14062500000000	fi(pi,1,16,8)	0.0308
3.14062500000000	fi(pi,1,16,9)	0.0308
3.14160156250000	fi(pi,1,16,10)	-0.0003
3.14160156250000	fi(pi,1,16,11)	-0.0003
3.14160156250000	fi(pi,1,16,12)	-0.0003
3.14160156250000	fi(pi,1,16,13)	-0.0003
1.99993896484375	fi(pi,1,16,14)	36.3400
3.14160156250000	fi(pi,0,16,14)	-0.0003
1.99996948242187	fi(pi,0,16,15)	36.3390

A forma estática de conversão tem pouca visibilidade. O grande problema se concentra no comportamento do controlador à medida que ele recebe valores não precisos devido ao limite de precisão numérica bem como seus cálculos também possuem erros devido aos mesmos limites de *hardware.*

Mediante isso, foi realizada uma bateria de testes sobre o comportamento de uma função matemática a medida que seus dados vão perdendo a precisão. A função escolhida foi $y = \pi^2 + \pi$ e ela foi implementada com 40 possibilidades com uma variação de precisão de 12 bits, totalizando 480 testes e resultados. Esses resultados são mostrados nas quatro tabelas seguintes onde é

apresentado o método de arredondamento, o método de overflow, o método de soma, os valores alcançados, os erros quando comparado ao valor Double e os bits de precisão.

Tabela 2 - Teste 01

Round Mode		Convergent		Fix		Nearest		Floor		Round	
Overflow mode		wrap		wrap		wrap		wrap		wrap	
Prod.&Sum Mode		Fullprecision		Fullprecision		Fullprecision		Fullprecision		Fullprecision	
BITS	**Valor real**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**
12:0	13.0112	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717
12:1	13.0112	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717
12:2	13.0112	13.8125	-6.1586	13.8125	-6.1586	13.8125	-6.1586	13.8125	-6.1586	13.8125	-6.1586
12:3	13.0112	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267
12:4	13.0112	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267
12:5	13.0112	13.1182	-0.8221	13.1182	-0.8221	13.1182	-0.8221	13.1182	-0.8221	13.1182	-0.8221
12:6	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:7	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:8	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:9	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:10	13.0112	5.9951	53.9234	5.9951	53.9234	5.9951	53.9234	5.9951	53.9234	5.9951	53.9234
12:11	13.0112	1.9985	84.6399	1.9985	84.6399	1.9985	84.6399	1.9985	84.6399	1.9985	84.6399
12:12	13.0112	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395

Round Mode		Convergent		Fix		Nearest		Floor		Round	
Overflow mode		Saturate		Saturate		Saturate		Saturate		Saturate	
Prod.&Sum Mode		FullPrecision		Fullprecision		Fullprecision		Fullprecision		Fullprecision	
BITS	**Valor real**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**
12:0	13.0112	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717
12:1	13.0112	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717
12:2	13.0112	13.8125	-6.1586	13.8125	-6.1586	13.8125	-6.1586	13.8125	-6.1586	13.8125	-6.1586
12:3	13.0112	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267
12:4	13.0112	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267
12:5	13.0112	13.1182	-0.8221	13.1182	-0.8221	13.1182	-0.8221	13.1182	-0.8221	13.1182	-0.8221
12:6	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:7	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:8	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:9	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:10	13.0112	5.9951	53.9234	5.9951	53.9234	5.9951	53.9234	5.9951	53.9234	5.9951	53.9234
12:11	13.0112	1.9985	84.6399	1.9985	84.6399	1.9985	84.6399	1.9985	84.6399	1.9985	84.6399
12:12	13.0112	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395

Tabela 3 - Teste 02

Round Mode		Convergent		Fix		Nearest		Floor		Round	
Overflow mode		Saturate		Saturate		Saturate		Saturate		Saturate	
Prod.&Sum Mode		KeepMSB		KeepMSB		KeepMSB		KeepMSB		KeepMSB	
BITS	**Valor real**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**
12:0	13.0112	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717
12:1	13.0112	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717
12:2	13.0112	13.8125	-6.1586	13.8125	-6.1586	13.8125	-6.1586	13.8125	-6.1586	13.8125	-6.1586
12:3	13.0112	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267
12:4	13.0112	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267
12:5	13.0112	13.1182	-0.8221	13.1182	-0.8221	13.1182	-0.8221	13.1182	-0.8221	13.1182	-0.8221
12:6	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:7	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:8	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:9	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:10	13.0112	5.9951	53.9234	5.9951	53.9234	5.9951	53.9234	5.9951	53.9234	5.9951	53.9234
12:11	13.0112	1.9985	84.6399	1.9985	84.6399	1.9985	84.6399	1.9985	84.6399	1.9985	84.6399
12:12	13.0112	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395

Round Mode		Convergent		Fix		Nearest		Floor		Round	
Overflow mode		Wrap		Wrap		Wrap		Wrap		Wrap	
Prod.&Sum Mode		KeepMSB		KeepMSB		KeepMSB		KeepMSB		KeepMSB	
BITS	**Valor real**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**
12:0	13.0112	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717
12:1	13.0112	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717
12:2	13.0112	13.8125	-6.1586	13.8125	-6.1586	13.8125	-6.1586	13.8125	-6.1586	13.8125	-6.1586
12:3	13.0112	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267
12:4	13.0112	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267
12:5	13.0112	13.1182	-0.8221	13.1182	-0.8221	13.1182	-0.8221	13.1182	-0.8221	13.1182	-0.8221
12:6	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:7	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:8	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:9	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:10	13.0112	5.9951	53.9234	5.9951	53.9234	5.9951	53.9234	5.9951	53.9234	5.9951	53.9234
12:11	13.0112	1.9985	84.6399	1.9985	84.6399	1.9985	84.6399	1.9985	84.6399	1.9985	84.6399
12:12	13.0112	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395

Tabela 4 - Teste 03

Round Mode		Convergent		Fix		Nearest		Floor		Round	
Overflow mode		Wrap		Wrap		Wrap		Wrap		Wrap	
Prod.&Sum Mode		KeepLSB		KeepLSB		KeepLSB		KeepLSB		KeepLSB	
BITS	**Valor real**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**
12:0	13.0112	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717
12:1	13.0112	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717
12:2	13.0112	13.8125	-6.1586	13.8125	-6.1586	13.8125	-6.1586	13.8125	-6.1586	13.8125	-6.1586
12:3	13.0112	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267
12:4	13.0112	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267
12:5	13.0112	13.1182	-0.8221	13.1182	-0.8221	13.1182	-0.8221	13.1182	-0.8221	13.1182	-0.8221
12:6	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:7	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:8	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:9	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:10	13.0112	5.9951	53.9234	5.9951	53.9234	5.9951	53.9234	5.9951	53.9234	5.9951	53.9234
12:11	13.0112	1.9985	84.6399	1.9985	84.6399	1.9985	84.6399	1.9985	84.6399	1.9985	84.6399
12:12	13.0112	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395

Round Mode		Convergent		Fix		Nearest		Floor		Round	
Overflow mode		Saturate		Saturate		Saturate		Saturate		Saturate	
Prod.&Sum Mode		KeepLSB		KeepLSB		KeepLSB		KeepLSB		KeepLSB	
BITS	**Valor real**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**
12:0	13.0112	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717
12:1	13.0112	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717
12:2	13.0112	13.8125	-6.1586	13.8125	-6.1586	13.8125	-6.1586	13.8125	-6.1586	13.8125	-6.1586
12:3	13.0112	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267
12:4	13.0112	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267	12.8906	0.9267
12:5	13.0112	13.1182	-0.8221	13.1182	-0.8221	13.1182	-0.8221	13.1182	-0.8221	13.1182	-0.8221
12:6	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:7	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:8	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:9	13.0112	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542	13.0042	0.0542
12:10	13.0112	5.9951	53.9234	5.9951	53.9234	5.9951	53.9234	5.9951	53.9234	5.9951	53.9234
12:11	13.0112	1.9985	84.6399	1.9985	84.6399	1.9985	84.6399	1.9985	84.6399	1.9985	84.6399
12:12	13.0112	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395

Tabela 5 - Teste 04

Round Mode		Convergent		Fix		Nearest		Floor		Round	
Overflow mode		Saturate		Saturate		Saturate		Saturate		Saturate	
Prod.&Sum Mode		SpecifyPrecision		SpecifyPrecision		SpecifyPrecision		SpecifyPrecision		SpecifyPrecision	
BITS	**Valor real**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**
12:0	13.0112	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717
12:1	13.0112	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717
12:2	13.0112	13.7500	-5.6782	13.7500	-5.6782	13.8750	-6.6389	13.7500	-5.6782	13.8750	-6.6389
12:3	13.0112	12.8750	1.0468	12.8750	1.0468	12.8750	1.0468	12.8750	1.0468	12.8750	1.0468
12:4	13.0112	12.8750	1.0468	12.8750	1.0468	12.9063	0.8066	12.8750	1.0468	12.9063	0.8066
12:5	13.0112	13.1250	-0.8747	13.1094	-0.7546	13.1250	-0.8747	13.1094	-0.7546	13.1250	-0.8747
12:6	13.0112	13.0078	0.0260	13.0000	0.0861	13.0078	0.0260	13.0000	0.0861	13.0078	0.0260
12:7	13.0112	11.1367	14.4067	11.1367	14.4067	11.1367	14.4067	11.1367	14.4067	11.1367	14.4067
12:8	13.0112	7.1387	45.1344	7.1387	45.1344	7.1387	45.1344	7.1387	45.1344	7.1387	45.1344
12:9	13.0112	5.1396	60.4983	5.1396	60.4983	5.1396	60.4983	5.1396	60.4983	5.1396	60.4983
12:10	13.0112	2.9985	76.9542	2.9985	76.9542	2.9985	76.9542	2.9985	76.9542	2.9985	76.9542
12:11	13.0112	1.4993	88.4771	1.4993	88.4771	1.4993	88.4771	1.4993	88.4771	1.4993	88.4771
12:12	13.0112	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395

Round Mode		Convergent		Fix		Nearest		Floor		Round	
Overflow mode		Wrap		Wrap		Wrap		Wrap		Wrap	
Prod.&Sum Mode		SpecifyPrecision		SpecifyPrecision		SpecifyPrecision		SpecifyPrecision		SpecifyPrecision	
BITS	**Valor real**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**
12:0	13.0112	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717
12:1	13.0112	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717	12.0000	7.7717
12:2	13.0112	13.7500	-5.6782	13.7500	-5.6782	13.8750	-6.6389	13.7500	-5.6782	13.8750	-6.6389
12:3	13.0112	12.8750	1.0468	12.8750	1.0468	12.8750	1.0468	12.8750	1.0468	12.8750	1.0468
12:4	13.0112	12.8750	1.0468	12.8750	1.0468	12.9063	0.8066	12.8750	1.0468	12.9063	0.8066
12:5	13.0112	13.1250	-0.8747	13.1094	-0.7546	13.1250	-0.8747	13.1094	-0.7546	13.1250	-0.8747
12:6	13.0112	13.0078	0.0260	13.0000	0.0861	13.0078	0.0260	13.0000	0.0861	13.0078	0.0260
12:7	13.0112	-2.9961	123.0270	-2.9961	123.0270	-2.9961	123.0270	-2.9961	123.0270	-2.9961	123.0270
12:8	13.0112	5.0039	61.5415	5.0039	61.5415	5.0039	61.5415	5.0039	61.5415	5.0039	61.5415
12:9	13.0112	5.0039	61.5415	5.0039	61.5415	5.0039	61.5415	5.0039	61.5415	5.0039	61.5415
12:10	13.0112	1.9951	84.6662	1.9951	84.6662	1.9951	84.6662	1.9951	84.6662	1.9951	84.6662
12:11	13.0112	0.9985	92.3256	0.9985	92.3256	0.9985	92.3256	0.9985	92.3256	0.9985	92.3256
12:12	13.0112	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395	0.7495	94.2395

Conclusão

Como conclusão prévia pode-se indicar que o modo de overflow ativo chamado *Wrap* somado a forma de operação *FullPrecision* com a forma de truncamento chamada *Floor* apresentam valores muito próximos ou idênticos aos outros encontrados quando comparados dentro de uma faixa de trabalho média, como por exemplo de 12:4 até 12:9. Em especial o último caso analisado com forma de soma e produto chamado SpecifyPrecision foi o teste que apresentou os menores erros, mas isso se deve ao formato de cálculo onde ele possui o dobro de bits de precisão para soma e produto quando comparado aos outros testes, ou seja, o maior problema nas operações não está no valor das constantes ou valores utilizados mas sim no tamanhos dos acumuladores e memórias que receberam esses valores.

APÊNDICE C – Operações matemáticas em FPGA com o VHDL 2008

OPERAÇÕES MATEMÁTICAS EM FPGA COM O VHDL 2008

Durante os testes iniciais dessa dissertação para provar o funcionamento dos cálculos numéricos em FPGA usando ponto fixo do VHDL 2008 foram utilizadas funções e operações matemáticas mais básicas para demonstrar possíveis problemas encontrados para só então partirmos para os controladores que são elementos mais complexos.

Os primeiros testes foram com as operações matemáticas de soma, multiplicação, subtração, divisão normal e divisão pelo recíproco (multiplicação pelo inverso do denominador).

Os dados de entrada foram apenas os números A e B que foram calculados e redimensionados em FPGA.

Os dados alcançados estão na tabela 1, logo abaixo, onde os cálculos foram realizados com 5 bits antes da vírgula, 5 bits depois da vírgula (05:05) e os números possuem sinalização, ou seja, podem aceitar valores negativos.

Tabela 1 – Comparação entre operação básicas em hardware com o comando resize e os valores esperados (simulados).

Entrada		**Valor Real (FPGA)**					**Valor Teórico**				
A	B	C=a+b	D=a*b	E=a-b	F=a/b	G=a*r(b)	C=a+b	D=a*b	E=a-b	F=a/b	G=a*r(b)
0	0	0.00	0.00	0.00	+ inf	0.00	0	0	0	+ inf	0
0	1	1.00	0.00	-1.00	0.00	0.00	1	0	-1	0	0
1	1	2.00	1.00	0.00	1.00	1.00	2	1	0	1	1
3	2	5.00	6.00	1.00	1.50	1.50	5	6	1	1.5	1.5
2	2	4.00	4.00	0.00	1.00	1.00	4	4	0	1	1
4.5	4.5	9.00	20.25	0.00	1.00	1.00	9	20.25	0	1	1
0	-0	0.00	0.00	0.00	+ inf	0.00	0	0	0	- inf	0
0	-1	-1.00	0.00	1.00	0.00	0.00	-1	0	1	0	0
1	-1	0.00	-1.00	2.00	1.00	-1.00	0	-1	2	-1	-1
3	-2	1.00	-6.00	5.00	1.50	-1.50	1	-6	5	-1.5	-1.5
2	-2	0.00	-4.00	4.00	1.00	-1.00	0	-4	4	-1	-1
-4.5	4.5	0.00	-20.25	-9.00	1.00	-1.00	0	-20.25	-9	-1	-1
-5	-5	-10.00	25.00	0.00	1.00	0.94	-10	25	0	1	1
-3	2	-1.00	-6.00	-5.00	1.50	-1.50	-1	-6	-5	-1.5	-1.5
-10	2	-8.00	-20.00	-12.00	5.00	-5.00	-8	-20	-12	-5	-5
1	4	5.00	4.00	-3.00	0.25	0.25	5	4	-3	0.25	0.25
-5	-2	-7	10	-3	2.5	2.5	-7	10	-3	2.5	2.5

Fonte: O Autor

De forma imediata é possível constatar que as divisões envolvendo números negativos ao ser utilizado o comando “resize” de redimensionar os dados tem uma forte tendência a apresentar erros principalmente de sinal, mas podendo apresentar também um pequeno erro em seu valor absoluto. Entre as duas formas de divisão, direta e pelo recíproco, a segunda possibilidade erros menores, principalmente quando observasse o sinal da resposta.

Para verificar a precisão alcançada foi utilizada a função indicada no apêndice “b” que é $y = \pi^2 + \pi$ com a mesma variação de bits de precisão.

A tabela 2 apresenta os valores alcançados com a compilação no software Quartus II e sua simulação no *Waveform*.

Tabela 2 – Comparação com a resposta encontrada no Quartus II com algumas simulações no Matlab.

Round Mode		**Quartus II (Resize)**	Fixed_round-Q		Round		Round		Round		Round	
Overflow mode			Saturate-Q		Saturate		Saturate		Saturate		Saturate	
Prod.&Sum Mode			Resize-Q		Fullprecision		KeepMSB		KeepLSB		SpecifyPrecision	
BITS	**Valor real**		**Erro**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**	**Resultado**	**E%**
12:0	13.0112	12.0000	1.0112	7.7717	12.000	0.000	12.000	0.000	12.000	0.000	12.000	0.000
12:1	13.0112	12.0000	1.0112	7.7717	12.000	0.000	12.000	0.000	12.000	0.000	12.000	0.000
12:2	13.0112	12.0000	1.0112	7.7717	13.813	-15.104	13.813	-15.104	13.813	-15.104	13.875	-15.625
12:3	13.0112	12.8750	0.1362	1.0468	12.891	-0.121	12.891	-0.121	12.891	-0.121	12.875	0.000
12:4	13.0112	12.8750	0.1362	1.0468	12.891	-0.121	12.891	-0.121	12.891	-0.121	12.906	-0.243
12:5	13.0112	12.8750	0.1362	1.0468	13.118	-1.889	13.118	-1.889	13.118	-1.889	13.125	-1.942
12:6	13.0112	13.0000	0.0112	0.0861	13.004	-0.032	13.004	-0.032	13.004	-0.032	13.008	-0.060
12:7	13.0112	13.0078	0.0034	0.0260	13.004	0.028	13.004	0.028	13.004	0.028	11.137	14.384
12:8	13.0112	13.0039	0.0073	0.0560	13.004	-0.002	13.004	-0.002	13.004	-0.002	7.139	45.104
12:9	13.0112	7.9980	5.0132	38.5295	13.004	-62.592	13.004	-62.592	13.004	-62.592	5.140	35.739
12:10	13.0112	3.9990	9.0122	69.2648	5.995	-49.915	5.995	-49.915	5.995	-49.915	2.999	25.018
12:11	13.0112	1.9995	11.0117	84.6324	1.999	0.049	1.999	0.049	1.999	0.049	1.499	25.018
12:12	13.0112	0.9998	12.0114	92.3162	0.750	25.031	0.750	25.031	0.750	25.031	0.750	25.031

Fonte: O Autor

Em especial as linhas 12:9, 12:10 e 12:11 demonstram o momento do limite de precisão dos dados e a comparação entre o real e os valores simulados. É verificável que os dados simulados no Matlab possuem uma tendência a apresentar valores mais próximos do real saindo da restrição imposta pelo número de bits, por exemplo, um número com 3 bits na sua parte inteira pode representar no máximo o número decimal 7 e em quase todas as simulações os valores alcançados estavam acima do limite. Além desse problema a última simulação do tipo SpecifyPrecision apresentou um valor abaixo do máximo (7) sendo 5, ou seja, ele calculou errado e apresentou um número abaixo do mais próximo possível que ele poderia representar.

CONCLUSÃO

Como conclusão prévia pode-se afirmar que as simulações possuem erros no caso de cálculos estáticos no limite da precisão dos dados entre inteiros e fracionários. Somado a isso, a divisão envolvendo números negativos apresenta um risco por inibir em seu valor final o sinal negativo de forma constante, além de alguns erros de cálculo de forma esporádica.

De forma direta na aplicação de controladores é importante tentar ficar longe da região limite de precisão e também diminuir a quantidade de operações que envolvem a divisão com números negativos.

É importante tentar ver o comportamento das operações matemáticas no caso de sistemas com funções ativas como são os controladores, pois estes tentam corrigir os erros gerados em cálculos passados caso isso afete o comportamento da planta que está sendo controlada.

APÊNDICE D - Gráficos e tabelas dos testes com baixa precisão nos controladores PID com plantas

GRÁFICOS E TABELAS DOS TESTES COM BAIXA PRECISÃO NOS CONTROLADORES

A quantidade de dados alocados em tabelas e na forma de gráficos gerados para provar o funcionamento obrigou a criação de mais que um apêndice nesta dissertação. Neste apêndice são apresentadas as tabelas com o controlador agindo sobre cada planta a medida que a precisão dos cálculos é degradada pela perda de bits, ou seja, casas decimais.

Nas tabelas são indicados os seguintes itens:

- Nº: número do teste;
- Precisão: é a quantidade de bits de precisão da parte inteira seguida da parte fracionada;
- IAES: índice de erro absoluto de um sistema "perfeito", é a referência gerada via software no Matlab;
- IAEC: índice de erro do sinal gerado pelo circuito;
- Erro%: esse é o erro entre o IAES e o IAEC mostrado em porcentagem;
- E.L: essa é a indicação da quantidade de unidades lógicas utilizadas na síntese do hardware.

Todos os gráficos são compostos por três comportamentos, são eles: malha aberta (MA), malha fechada (MF) e VHDL que é o comportamento em malha fechada em VHDL.

1. Planta 01 - Forno

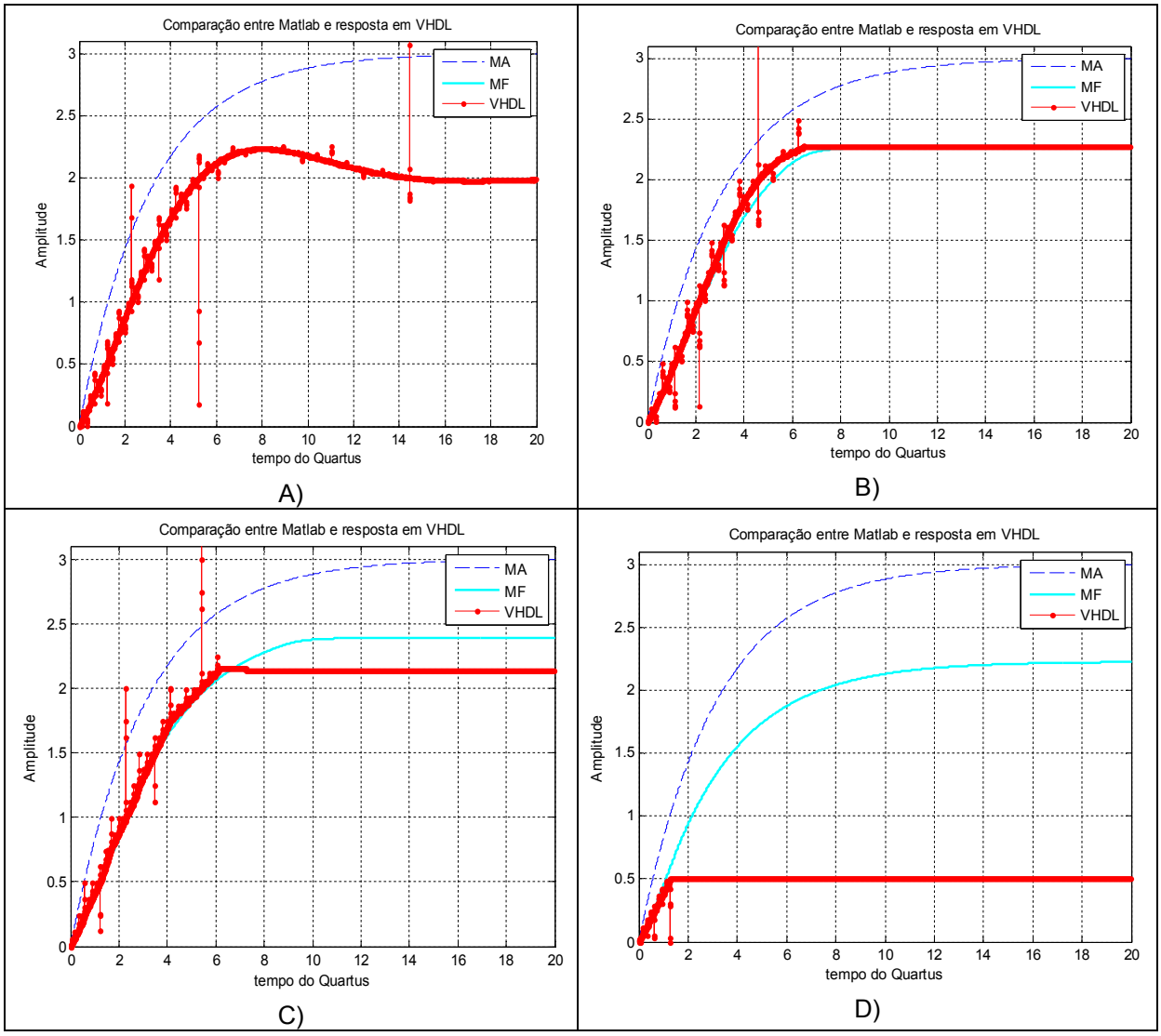
Os dados apresentados abaixo são as respostas do modelo do forno operando com uma amostragem de 0.01 segundo durante 20 segundos.

Nº	precisão	IAES	IAEC	erro%	E.L.
1	8:23	0.3008	0.3296	9.55	1371
2	8:22	0.3008	0.3251	8.08	1337
3	8:21	0.3008	0.3331	10.72	1286
4	8:20	0.3008	0.3331	10.74	1204
5	8:19	0.3008	0.3322	10.43	1157
6	8:18	0.3008	0.3366	11.89	1034
7	8:17	0.3008	0.3378	12.27	949
8	8:16	0.3008	0.3370	12.04	906
9	8:15	0.3008	0.3398	12.94	856
10	8:14	0.3008	0.3395	12.86	814
11	8:13	0.3008	0.3486	15.88	771
12	8:12	0.3009	0.3378	12.27	723
13	8:11	0.2992	0.3335	11.47	678
14	8:10	0.4185	0.4541	8.52	626
15	8:9	0.4912	0.37	-24.20	465
16	8:8	0.3668	1.52	313.434	440

Dentro dos testes realizados existiram 4 momentos que devem ser comentados. Esses momentos são comentados e as suas respectivas figuras são apresentadas abaixo.

- A) a simulação foi feita com a precisão 8:23 e o erro do ponto de vista gráfico foi inexistente;

- B) a simulação foi feita com a precisão de 8:10 e nesse momento apareceu um erro em regime permanente;
- C) a simulação foi feita com a precisão 8:09 e houve o aparecimento de um erro em regime permanente alto posicionado na parte superior da linha de referência.
- D) a simulação foi feita com a precisão 8:08 nesse momento o controlador e a planta passam do limite mínimo de precisão e já não mais representão o comportamento do sistema.



2. Planta 02 – Servomotor com carga

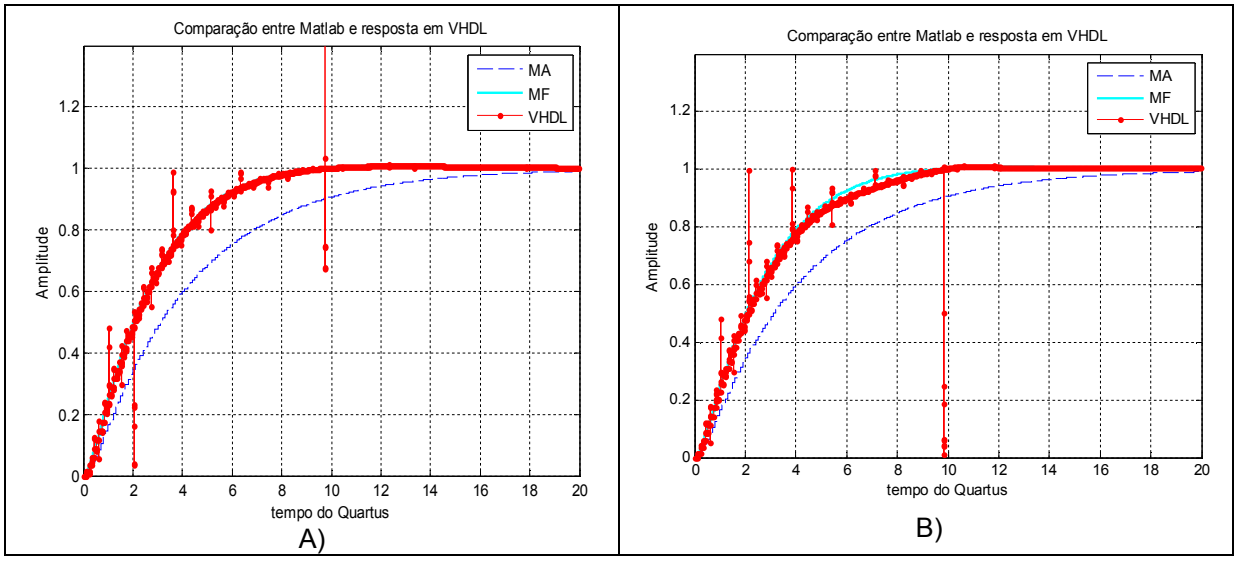
Os dados apresentados abaixo são as respostas do modelo do servomotor operando com uma amostragem de 0.1 segundo durante 20 segundos.

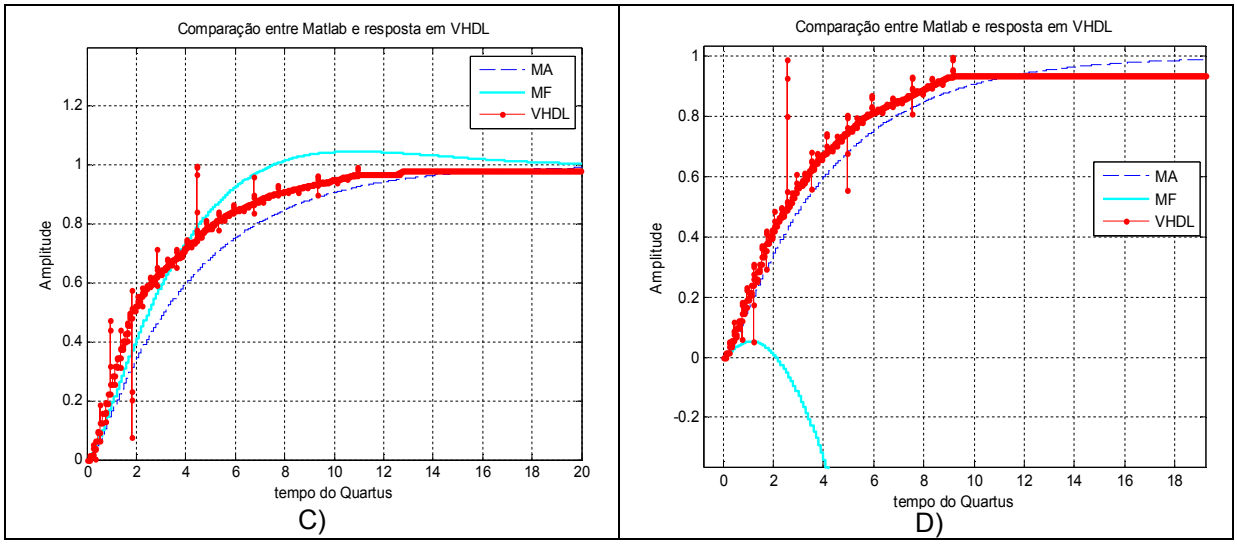
Nº	precisão	IAES	IAEC	erro%	E.L.
1	8:23	0.1310	0.1489	13.66	2010
2	8:22	0.1310	0.1485	13.34	1958
3	8:21	0.1310	0.1474	12.46	1882
4	8:20	0.1310	0.1455	11.02	1818

5	8:19	0.1310	0.1491	13.77	1727
6	8:18	0.1310	0.1492	13.85	1634
7	8:17	0.1310	0.1530	16.76	1505
8	8:16	0.1310	0.1555	18.66	1408
9	8:15	0.1310	0.1529	16.70	1290
10	8:14	0.1311	0.1564	19.35	1224
11	8:13	0.1309	0.1527	16.64	1157
12	8:12	0.1311	0.1630	24.27	1083
13	8:11	0.1308	0.1683	28.67	1019
14	8:10	0.1326	0.1629	22.83	943
15	8:9	0.1581	0.18	14.94	818
16	8:8	8.4562	0.23	-97.339	772

Dentro dos testes realizados existiram 4 momentos que devem ser comentados. Esses momentos são comentados e as suas respectivas figuras são apresentadas abaixo.

- A) a simulação foi feita com a precisão 8:23 e o erro do ponto de vista gráfico foi inexistente;
- B) a simulação foi feita com a precisão de 8:11 e nesse momento apareceu um erro em regime transitório;
- C) a simulação foi feita com a precisão 8:09 e houve o aparecimento de um erro em regime transitório diferente entre o simulado no Matlab em comparação com simulado em FPGA.
- D) a simulação foi feita com a precisão 8:08 e nesse momento o controlador simulado no Matlab gerou uma instabilidade fazendo com a continuação da análise fosse interrompida.





3. Planta 03 – Braço de Robô

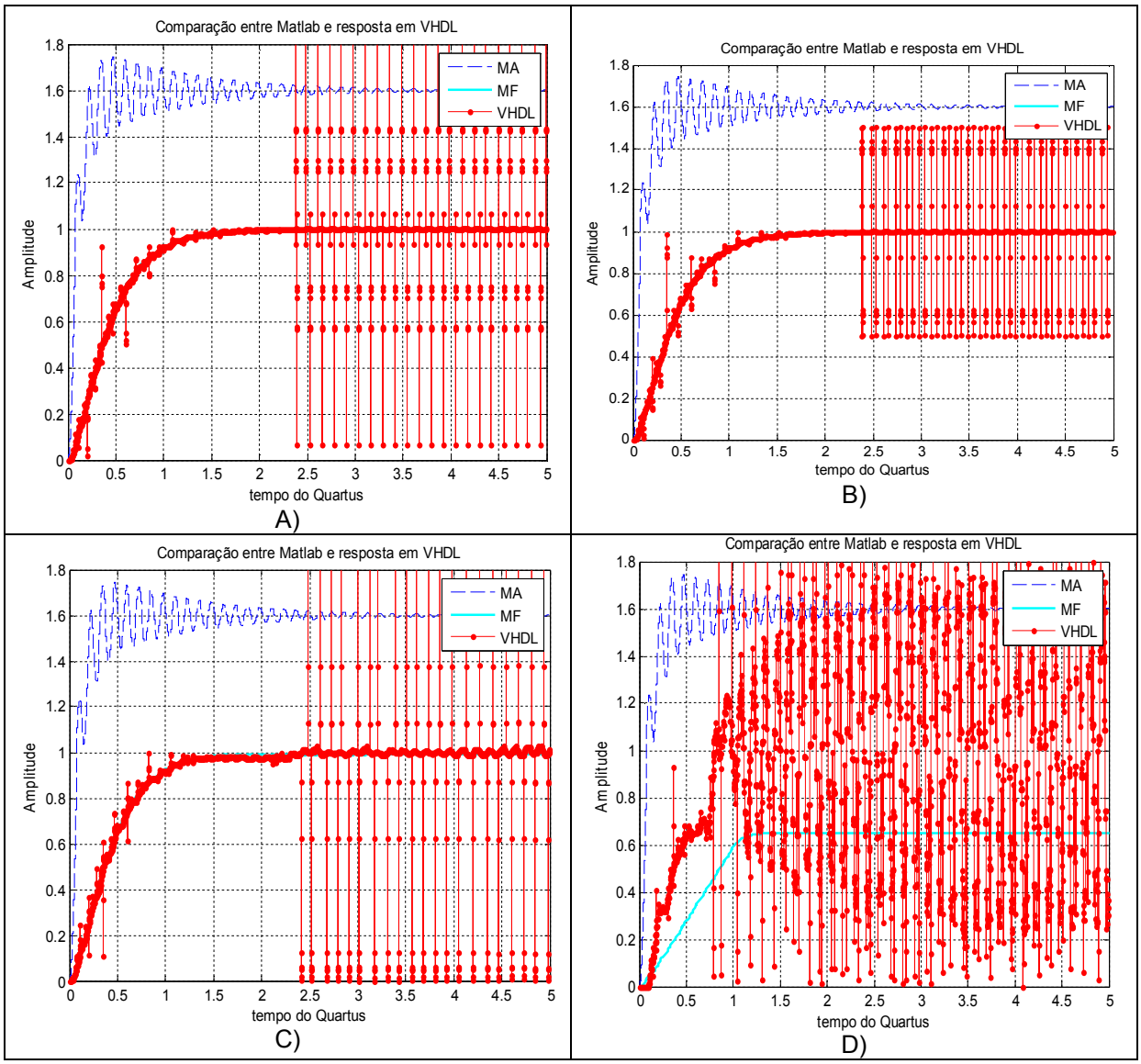
Os dados apresentados abaixo são as respostas do modelo do braço robótico operando com uma amostragem de 0.01 segundo durante 5 segundos.

Nº	precisão	IAES	IAEC	erro%	E.L.
1	8:23	0.0906	0.1296	43.11	4148
2	8:22	0.0906	0.1133	25.15	3873
3	8:21	0.0906	0.1345	48.54	3592
4	8:20	0.0906	0.1323	46.09	3225
5	8:19	0.0906	0.1301	43.70	2994
6	8:18	0.0906	0.1440	59.02	2572
7	8:17	0.0906	0.1457	60.84	2356
8	8:16	0.0905	0.1376	51.94	2235
9	8:15	0.0906	0.1284	41.78	2037
10	8:14	0.0908	0.1502	65.47	1927
11	8:13	0.0914	0.1452	58.90	1819
12	8:12	0.0932	0.1282	37.55	1703
13	8:11	0.0968	0.1506	55.69	1597
14	8:10	0.1062	0.1519	43.05	1462
15	8:9	0.1665	0.11	-34.04	1271
16	8:8	0.4255	0.42	-2.448	1197

Dentro dos testes realizados existiu 4 momentos que devem ser comentados. Esses momentos são comentados e as suas respectivas figuras são apresentadas abaixo.

- A) a simulação foi feita com a precisão 8:23 e o erro do ponto de vista gráfico foi inexistente;
- B) a simulação foi feita com a precisão de 8:20 e nesse momento as amplitudes dos valores intermediários do cálculo em hardware diminuiram de amplitude. Esse fato não comprometeu o comportamento dinâmico ou o regime da planta;
- C) a simulação foi feita com a precisão 8:11 e nesse momento é possível ver a planta mantendo pequenas oscilações durante o período de regime permanente. Na área de controle é descrito esse comportamento como sendo um controle marginalmente estável, o que para alguns autores, como o desta dissertação, esse já é um indício de instabilidade;

- D) a simulação foi feita com a precisão 8:08 e nesse momento a falta de precisão tanto da planta como do controlador geraram uma simulação descaracterizada, muito provavelmente pela natureza oscilatória da planta.



4. Planta 04 – Pêndulo

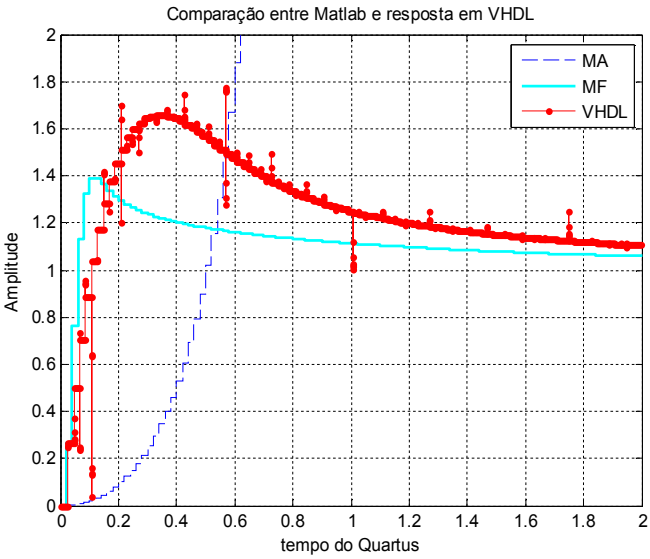
Os dados abaixo apresentados são as respostas do modelo do pêndulo invertido operando com uma amostragem de 0.002 segundo durante 2 segundos.

Nº	precisão	IAES	IAEC	erro%	E.L.
1	15:52	0.1515	0.3121	106.04	7683
2	15:51	0.1515	100.0000	65926.78	7600

Esse modelo estudado é naturamente instável e não linear, ou seja, duas características de difícil controle. Para tentar melhorar tanto o controle quanto a simulação foi aumentada a quantidade

de bits de precisão tanto da parte inteira quanto da parte fracionada, mas mesmo assim o comportamento não foi adequado para o controle.

O principal motivo para isso pode ser a grande variação existente entre a amplitude dos números. As constantes da planta e do controlador são baixas, mas a resposta do controlador trabalha com valores grandes. De forma direta esse modelo demonstra o limite da aplicação do ponto fixo em sistema de controle com ponto fixo.



Quando analisado o comportamento da planta em malha aberta não houve constatação de discrepância numérica ou gráfica, ficando então a cargo do controlador e do sinal de controle as limitações de trabalho com esse sistema.

CONCLUSÃO: para sistemas com pequena variação de suas amplitudes, sejam elas constantes, sinais de entrada e saída ou variáveis auxiliares a técnica do ponto fixo em *hardware* funciona, mas para sistemas com grande variação de amplitude essa técnica torna-se ineficaz e mesmo aumentando a quantidade de bits de precisão os erros, oscilações e discrepâncias são visualizadas.
Além disso, não é possível afirmar que o Matlab e o VHDL vão mostrar exatamente o mesmo tipo de erro, apenas que para casos bem resolvidos onde a precisão dos bits não está no limiar eles possuem comportamentos iguais e coerentes, a mesma análise é comprovada pelo apêndice C.