Universidade Federal do Amazonas

Faculdade de Tecnologia

Programa de Pós-Graduação em Engenharia Elétrica

Verificação de Modelos Aplicada ao Projeto de Controladores Digitais Implementados em Processadores de Ponto-Fixo

Hussama Ibrahim Ismail

Manaus – Amazonas

Novembro de 2015

Hussama Ibrahim Ismail

Verificação de Modelos Aplicada ao Projeto de Controladores Digitais Implementados em Processadores de Ponto-Fixo

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica, como requisito parcial para obtenção do Título de Mestre em Engenharia Elétrica. Área de concentração: Automação e Controle.

Orientador: Lucas Carvalho Cordeiro

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Ismail, Hussama Ibrahim
I83v Verificação de Modelos Aplicada ao Projeto de Controladores Digitais Implementados em Processadores de Ponto-Fixo / Hussama Ibrahim Ismail. 2015
95 f.: il. color; 31 cm.

Orientador: Lucas Carvalho Cordeiro
Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal do Amazonas.

1. Controladores Digitais em Ponto-Fixo. 2. Formas Diretas e Delta. 3. Métodos Formais. 4. Verificação Limitada de Modelos. I. Cordeiro, Lucas Carvalho II. Universidade Federal do Amazonas III. Título

HUSSAMA IBRAHIM ISMAIL

VERIFICAÇÃO DE MODELOS APLICADA AO PROJETO DE CONTROLADORES DIGITAIS IMPLEMENTADOS EM PROCESSADORES DE PONTO-FIXO.

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Amazonas, como requisito parcial para obtenção do título de Mestre em Engenharia Elétrica na área de concentração Controle e Automação de Sistemas.

Aprovado em 11 de novembro de 2015.

BANCA EXAMINADORA

Prof. Dr. Lucas Carvalho Cordeiro, Presidente

Universidade Federal do Amazonas- UFAM

Prof. Dr. Eddie Batista de Lima Filho, Membro Interno

Universidade Federal do Amazonas- UFAM

Prof. Dr. Marcos Flávio Silveira Vasconcelos D'Angelo, Membro Externo

Universidade Estadual de Montes Claros- UNIMONTES

Agradecimentos

Em primeiro lugar, agradeço a Deus pela vida, saúde, e por tudo que Ele tem me dado.

Agradeço a minha família, em especial minha mãe, Maria Lúcia Pacheco Farias pelo encorajamento, motivação e compreensão durante mais essa etapa.

Agradeço ao professor Lucas Cordeiro pela oportunidade dada, por toda a sua orientação, pela paciência, pelo aprendizado e por todos os trabalhos feitos em conjunto. Você é um excelente profissional e um exemplo de excelente professor para todos nós!

Agradeço a todos os colegas de laboratório, alunos e professores, a qual tive a oportunidade de conviver durante o período de estudos. Em especial, ao companheiro de disciplinas e estudos Iury Bessa. Iury, muito obrigado pelos conselhos, pelos esclarecimentos, pela paciência, por todas as dicas e pela oportunidade de poder trabalhar com você. Este trabalho também é mérito seu. Tenho certeza que você irá longe!

Agradeço também a todas as pessoas que colaboram, ou que colaboraram, direta ou indiremente com o desenvolvimento deste projeto, posso citar: Eddie de Lima Filho, Iury Bessa, João Edgar Chaves Filho, Lucas Cordeiro, Mauro Freitas, Mikhail Ramalho, Renato Abreu e Waldir Sabino Jr.

Parte dos resultados apresentados neste trabalho foram obtidos através do projeto de pesquisa e formação de recursos humanos, à níveis de graduação e pós-graduação, nas áreas de automação industrial, software para dispositivos móveis e TV digital, patrocinado pela Samsung Eletrônica da Amazônia Ltda, nos termos da Lei Federal brasileira número 8.387/91.

"Mas, como está escrito: Nem olhos viram,
nem ouvidos ouviram, nem jamais penetrou em
coração humano o que Deus tem preparado
para aqueles que o amam."

1 Coríntios 2:9

Resumo

O uso extensivo de controladores digitais implementados em ponto-fixo demandam um maior esforço para prevenir erros de projeto que aparecem no domínio discreto. Este trabalho descreve uma nova metodologia de verificação que emprega verificação de modelos limitada baseada em satisfação booleana e teorias do módulo da satisfatibilidade, para verificar a ocorrência de erros de projetos em controladores digitais causados pelos efeitos da palavra finita. Neste trabalho, serão comparados os desempenhos das realizações na formas delta e as tradicionais formas diretas. Os resultados mostram que a forma delta reduz substancialmente a fragilidade de controladores digitais, se comparados com as formas diretas. A metodologia proposta é eficiente para verificar controladores digitais do mundo real. Ela foi conclusiva em aproximadamente 89% dos casos de teste.

Palavras-chave: controladores digitais em ponto-fixo; formas diretas e delta; métodos formais; verificação limitada de modelos.

Abstract

The extensive use of fixed-point digital controllers demands a growing effort to prevent design errors that appear in discrete-time domain. The present work describes a novel verification methodology, which employs bounded model checking based on boolean satisfiability and satisfiability modulo theories to verify the occurrence of design errors, due to the finite word-length format, in fixed-point digital controllers. Here, the performance of digital controllers realizations that use delta operators are compared to those that use traditional direct forms. Experimental results show that the delta-form realization substantially reduces the digital controllers' fragility when compared to the direct-form realization. Additionally, the proposed methodology is very effective and efficient to verify real-world digital controllers, where conclusive results are obtained in nearly 89% of the benchmarks.

Keywords: fixed-point digital controllers; direct and delta forms; formal methods; bounded model checking

Índice

Índice de Figuras

Índice de Tabelas

Abreviações

ANSI-C - *American National Standards Institute C Programming Language*

BIBO - *Bounded-Input Bounded-Output*

BMC - *Bounded Model Checking*

CBMC - *C Bounded Model Checker*

CFG - *Control Flow Graph*

CNF - *Conjunctive Normal Form*

DFI - *Direct Form I*

DFII - *Direct Form II*

DDFI - *Delta Direct Form I*

DDFII - *Delta Direct Form II*

DSP - *Digital Signal Processor*

DSVerifier - *Digital-Systems Verifier*

ESBMC - *Efficient SMT-Based Context-Bounded Model Checker*

FPGA - *Field Programable Gate Array*

FWL - *Finite Word Length*

GUI - *Graphical User Interface*

LCO - *Limit Cycle Oscilations*

LTI - *Linear Time-Invariant*

PWM - *Pulse Width Modulation*

RISC - *Reduced Instruction Set Computer*

SAT - *Boolean Satisfiability*

SISO - *Single-Input Single-Output*

SMT - *Satisfatibility Module Theories*

SSA - *Single Static Assignment*

SV-COMP - ***I**nternational **C**ompetition on **S**oftware **V**erification*

TDFII - ***T**ransposed **D**irect **F**orm **II***

TDDFII - ***T**ransposed **D**elta **D**irect **F**orm **II***

VC - ***V**erification **C**ondition*

WCET - ***W**orst-**C**ase **E**xecution **T**ime*

Capítulo 1

Introdução

Atualmente, a comunidade de controle tem utilizado controladores digitais devido às suas diversas vantagens em confiabilidade, sensibilidade, flexibilidade e custo sobre os controladores analógicos. No entanto, esses tipos de controladores também apresentam algumas desvantagens no seu uso, como, por exemplo, erros e arredondamentos indesejados durante o processo de quantização. Nesse contexto, existem várias técnicas afim de resolver problemas pertencentes ao domínio discreto, em especial problemas relacionados à palavra finita [1, 2].

Controladores digitais são geralmente utilizados em microcomputadores, microprocessadores, processadores digitais de sinais (*Digital Signal Processors*, DSP) [3] e em arranjo de portas programáveis em campo (*Field Programable Gate Array*, FPGA) [4]. De acordo com a escolha do *hardware*, o formato e aritmética utilizados para representar números podem mudar (*e.g.*, número de *bits*, aritmética de ponto-fixo ou flutuante). Tais representações influenciam diretamente a precisão e desempenho do controlador digital. De modo geral, processadores de ponto-flutuante possuem um número maior de valores representáveis e, consequentemente, uma melhor precisão. No entanto, processadores de ponto-fixo fornecem soluções mais baratas e rápidas, sendo assim, bastante utilizados na prática.

O cenário previamente mencionado necessita de um melhor entendimento dos problemas em implementações de controladores digitais, que são: quantização e palavra finita. Ambos podem ser minimizados durante o projeto do controlador. De fato, vários fatores podem amplificar ou atenuar esses efeitos, *e.g.*, o uso de realizações nas formas diretas ou delta, bem como a definição do número de *bits* e taxas de amostragem. Além disso, a possível influência desses efeitos pode trazer problemas de estabilidade e sensibilidade nos polos e zeros do controlador

digital. Esses efeitos já foram anteriormente investigados por vários autores [5–9].

Para evitar a degradação do desempenho, engenheiros de controle investem tempo e esforço na fase de projeto, resolvendo problemas relacionados aos efeitos de palavra finita com soluções mais robustas. Trabalhos anteriores propõem a utilização de métodos especiais, como o operador *delta* [6, 10]. Outros, desenvolveram diferentes metodologias para estimar valores ótimos do tamanho da palavra, com o intuito de mitigar seus efeitos [11–14].

Existem iniciativas que propõem controladores mais complexos, mantendo o seu desempenho dentro de limites de erro (*e.g.*, controladores robustos e não frágeis [15, 16]). Além disso, ferramentas de verificação automática também estão sendo aplicadas em sistemas de tempo discreto, afim de encontrar erros em projetos (*e.g.*, UPPAAL [17], Open-Kronos [18], CPN [19], e Maellan [20]). No entanto, ainda existe uma deficiência na verificação formal de sistemas embarcados, em especial para controladores digitais.

Diferente de outros estudos, este trabalho apresenta uma metodologia para verificar a ocorrência de erros nos projetos e implementações de controladores digitais, aplicando a verificação de modelos limitada (*Bounded Model Checking*, BMC). A metodologia permite verificar cinco propriedades em controladores digitais, *e.g.*, estouros aritméticos, ciclo limite, restrições temporais, estabilidade e fase mínima, incluindo também diferentes formas de realização. A propriedade de estouros aritméticos avalia se as saídas produzidas pelo controlador digital estão dentro do intervalo definido pela palavra finita. A propriedade de ciclo limite checa a presença de oscilações indesejadas na saída. Além disso, a verificação de restrições temporais compara o tempo de resposta do controlador digital com o *deadline* estabelecido. Por fim, as propriedades de estabilidade e fase mínima analisam o posicionamento dos polos e zeros do sistema.

1.1 Descrição do Problema

A escolha por processadores de ponto-fixo é motivada pelo baixo custo e velocidade. No entanto, o engenheiro de controle, durante o projeto, deve levar em consideração todas as suas características de *hardware*. Por isso, o problema a ser tratado por essa dissertação está relacionado à encontrar falhas de funcionamento dos controladores nas respectivas plataformas de *hardware*, que podem levar desde perdas financeiras, até a perda de vidas humanas.

Historicamente, podemos citar diversas falhas de funcionamento em dispositivos, que poderiam ser previstos durante a etapa de projeto. Por exemplo, o foguete Ariane 5, que em

1996, explodiu 40 segundos após o seu lançamento, por causa de um estouro arimético durante a conversão dos dados de 64-*bits* para 16-*bits*, não prevista no *software*, causando um prejuízo de 7 milhões de dólares [21]. Outro exemplo aconteceu com o interceptador de mísseis americano Patriot, que em 1991, durante a guerra no golfo, não foi capaz de detectar um míssil inimigo em virtude de erros de arredondamento, causados pela precisão finita. Este caso levou à morte de 28 militares americanos e feriu outros 98 [21].

1.2 Objetivos

O objetivo geral dessa dissertação é mostrar que a verificação de modelos limitada pode ser utilizada como uma técnica poderosa na validação do projeto de controladores digitais, auxiliando o engenheiro de controle através de uma abordagem eficiente de verificação, trazendo mais confiança e menos esforço, quando comparada com às tradicionais ferramentas de simulação (*e.g.*, Matlab [22] e LabVIEW [23]).

Os objetivos específicos são:

- Propor uma metodologia e respectiva ferramenta, para verificação de controladores digitais implementados em processadores de ponto-fixo, considerando efeitos de quantização e palavra finita;
- Desenvolver os algoritmos para realização nas formas delta, e comparar com as formas diretas;
- Desenvolver um algoritmo para verificação de estabilidade e fase mínima que seja suportado por um verificador de modelos, visando o menor custo computacional possível;
- Desenvolver um algoritmo para detecção de ciclo limite considerando entradas nulas e não determinísticas;
- Avaliar experimentalmente a metodologia na verificação de controladores digitais realizados com formas diretas e delta;
- Verificar a fragilidade e robustez de controladores digitais.

1.3 Contribuições

Este trabalho descreve três importantes contribuições. Primeiro, o estudo pioneiro em aplicar a técnica BMC, baseada em satisfação booleana, e teorias do módulo da satisfatibilidade, para verificar a ocorrência de erros de projeto relacionados à palavra finita e efeitos de quantização em controladores implementados em ponto-fixo. Segundo, demonstra formalmente que as realizações nas formas delta apresentam melhores resultados em propriedades importantes (*e.g.*, ciclo limite, estabilidade e fase mínima), se comparado às realizações nas formas diretas. Por último, apresenta uma metodologia de verificação que é eficiente para verificar controladores do mundo real.

1.4 Trabalhos Relacionados

Os efeitos da palavra finita, em controladores digitais, são bastante conhecidos na literatura de sistemas de controle. Grande parte dos pesquisadores de tais sistemas tentam prevenir esses problemas com esforços adicionais na fase de projeto, como será apontado no Capítulo 2. Esses esforços envolvem a implementação de controladores não frágeis e possuem um formato ótimo da palavra finita. Um panorama sobre o assunto pode ser visto em Istepanian e Whidborne [1].

Atualmente, poucos pesquisadores têm desenvolvido algoritmos para verificar a ocorrência de efeitos da palavra finita, uma vez que, a maioria dos trabalhos relacionados estão focados em métodos de como preveni-los. Contudo, tais algoritmos são importantes para validar e verificar a eficácia destes métodos. Com base nisso, testes e simulações são bastante utilizados durante a etapa de projeto de um controlador digital. Porém, tais técnicas não garantem a sua corretude, visto que exploram apenas um subconjunto dos possíveis comportamentos do sistema [24]. Um exemplo de ferramenta de simulação é proposta por Sung e Kum [13], onde os autores desenvolveram um método sistemático que determina o formato de palavra finita, na representação de ponto-fixo, mediante restrições de custo de *hardware* e ruídos decorrentes do processo de quantização. No entanto, assim como outras ferramentas de simulação, esta oferece uma validação parcial e não cobre todos os cenários. Como resultado, alguns problemas podem não ser detectados. Outro estudo interessante é apresentado por Anta *et al.* [25], onde uma ferramenta chamada Costan encontra erros na implementação de um modelo matemático e verifica

se esse erro é tolerado, considerando os efeitos de quantização e implementação em ponto-fixo. Os autores focam na verificação de estabilidade do sistema. Além disso, recentemente Luengo *et al.* [26] apresentaram uma ferramenta de simulação que é aprimorada com o algoritmo de Monte Carlo para detectar e prever oscilações de ciclo limite, em filtros digitais. Esse estudo é uma alternativa muito interessante em termos de eficiência e tempo. Adicionalmente, existem vários estudos relacionados ao desenvolvimento de ferramentas para avaliar o desempenho de controladores e filtros digitais, estimando ruído devido aos arredondamentos ocasionados pela palavra finita [27–32].

Alguns exemplos particulares de uso das técnicas BMC, em sistemas de controle, devem ser citados [33–35]. Dutertre *et al*. mostram as vantagens dos métodos formais sobre as tradicionais ferramentas de depuração [33]. Os autores usam uma ferramenta baseada em solucionadores SMT (*Satisfatibility Module Theories*) para diagnosticar e monitorar sistemas de aviação. Além disso, Simko e Jackson demonstraram que um controlador digital pode ser formalmente verificado com a combinação de SMT (para verificar o *software* de controle) e modelos de Taylor (para predizer a dinâmica da planta contínua) [34]. Por outro lado, Prabhu e Dasgupta [35] mostram a verificação de modelos aplicada à controladores discretos que reagem à eventos específicos, e que podem ser representados em uma máquina de estados finitos. Neste trabalho, os autores usam uma combinação de solucionadores SMT e ferramentas industriais de verificação de modelos.

Existem algumas ferramentas de verificação de modelos amplamente conhecidas e utilizadas para verificar sistemas de tempo real (*e.g.*, UPPAAL [17]). O UPPAAL é um verificador de modelos baseado na teoria de autômatos temporizados, utilizado para verificar sistemas de tempo real modelados por tais autômatos. Tal ferramenta tem vasta aplicação e tem sido utilizada com sucesso na verificação de protocolos de comunicação. Outra ferramenta similar é o Open-Kronos [18], que é capaz de checar a alcançabilidade dos autômatos temporizados. Além disso, a ferramenta CPN [19] também tem sido aplicada para verificar sistemas modelados com redes de Petri coloridas (com ou sem temporização). Por outro lado, trabalhos mais recentes utilizaram as técnicas BMC, baseadas em solucionadores SMT, para verificar propriedades de filtros e controladores digitais. Cox *et al.* [36, 37] mostram que ferramentas de simulação são úteis, mas insuficientes para detectar erros de projeto. Os autores propõem usar solucionadores SMT, alinhadas com a verificação de modelos limitada e ilimitada, para verificar a presença de estouros aritméticos e oscilações de ciclo limite em filtros digitais.

A ferramenta BMC, desenvolvida nesse trabalho, é chamada *Digital-Systems Verifier* (DSVerifier) [38]. Diferente de Costan que efetua uma análise estática do código do controlador, o DSVerifier efetua uma execução simbólica, que explora todos os possíveis comportamentos durante a sua realização, até uma determinada profundidade. E ao contrário das ferramentas UPPAAL, Open-Kronos e CPN, onde o usuário modela o autômato referente ao sistema digital, as suas restrições e propriedades, no DSVerifier este processo é automatizado pelo verificador de modelos limitados. Além disso, difere-se de Cox [36, 37], pois as implementações referentes à biblioteca de aritmética, em ponto-fixo, formas de realizações dos sistemas digitais e métodos para avaliações das propriedades são escritos em ANSI-C, ao invés de puramente satisfação booleana (*Boolean Satisfiability*, SAT) ou SMT (*i.e.*, em um nível maior de abstração). Tal característica permite que o DSVerifier seja conectado à diferentes verificadores de modelos, como o *C Bounded Model Checker* (CBMC) e o *Efficient SMT-Based Context-Bounded Model Checker* (ESBMC).

Anteriormente, Abreu *et al.* [39] aplicaram o DSVerifier para verificar estouros aritméticos, presença de ciclo limite, restrições temporais, estabilidade e resposta em frequência para filtros digitais utilizando o ESBMC. Mais recentemente, Bessa *et al.* [40] aplicaram as propriedades de estouro aritmético, ciclo limite e restrições de temporais para controladores digitais. Em geral, este trabalho é uma extensão de Abreu *et al.* [39] e Bessa *et al.* [40], onde a ferramenta é aplicada para verificar os efeitos da palavra finita, usando uma maior variedade de controladores, propriedades, formas de realizações e verificadores de modelos.

1.5 Organização da Dissertação

Neste capitulo, foram descritos o contexto, a motivação e trabalhos relacionados com o tema deste estudo. O Capítulo 2 descreve uma breve introdução sobre controladores digitais, abordando alguns tópicos relacionados à implementação em processadores de ponto-fixo, incluindo as realizações nas formas diretas e delta. Nele também são abordadas as teorias do módulo da satisfatibilidade em conjunto com a verificação de modelos limitada. Em seguida, o Capítulo 3 descreve a metodologia de verificação para controladores digitais, incluindo detalhes sobre a verificação de estouros aritméticos, ciclo limite, restrições temporais, estabilidade e fase mínima. O Capítulo 4 apresentará a avaliação experimental contendo a descrição dos casos de teste, configuração dos experimentos e os resultados obtidos por meio da ferramenta DSVeri-

fier. As considerações finais e ideias para trabalhos futuros serão apresentadas no Capítulo 5. Além disso, no Apêndice A, podem ser encontradas mais informações sobre a implementação e a utilização do DSVerifier. Por fim, os controladores digitais utilizados como casos de teste estão disponíveis no Apêndice B.

Capítulo 2

Fundamentação Teórica

Neste capítulo, conceitos sobre controladores digitais serão apresentados, levando em consideração as suas estruturas, realizações e os efeitos devido à utilização da palavra de comprimento finito. As realizações e as restrições inerentes à representação em ponto-fixo também serão apresentadas, juntamente com conceitos sobre verificação de modelos limitada e lógica de primeira ordem, envolvendo as teorias do módulo da satisfatibilidade.

2.1 Controladores Digitais

Um controlador digital pode ser definido como um sistema causal, linear, discreto e invariante no tempo [41], podendo ser representado matematicamente de várias maneiras (*e.g.*, funções de transferência, equações de espaço de estado e equações de diferença). Essas representações são estudadas em vários livros de sinais e sistemas (*e.g.*, [42, 43]). Uma representação importante para os controladores digitais é a equação de diferença com coeficientes constantes, a qual pode ser descrita como:

$$y(n) = -\sum_{k=1}^{N} a_k y(n-k) + \sum_{k=0}^{M} b_k x(n-k), \tag{2.1}$$

onde $y(n)$ é a saída e $x(n)$ é a entrada em uma n-ésima amostra [43]. Além disso, N é chamada de ordem da equação de diferença e descreve o número de recorrências nas saídas, M descreve o número de recorrências nas entradas, por fim a_k e b_k descrevem os coeficientes que operam sobre tais saídas e entradas [43].

Aplicando a transformada z na Equação (2.1) [42], o controlador digital pode ser representado pela seguinte função de transferência:

$$H(z) = \frac{b_0 + b_1 z^{-1} + ... + b_M z^{-M}}{1 + a_1 z^{-1} + ... + a_N z^{-N}}, \tag{2.2}$$

onde z é chamado de operador *forward-shift*, e z^{-1} é chamado de operador *backward-shift*.

2.2 Realização de Controladores Digitais

Um controlador digital descrito pela Equação (2.2) pode ser implementado em *software* de várias maneiras, por exemplo, utilizando formas diretas e delta. No entanto, em um ambiente com precisão finita, a escolha da estrutura de realização pode influenciar no desempenho devido aos efeitos da quantização. Em geral, tais efeitos são caracterizados pela aproximação de um valor de sinal para valores do conjunto discreto finito.

2.2.1 Formas Diretas

As formas diretas usam diretamente os coeficientes da Equação (2.1) na sua implementação. A sua principal vantagem, é que as variáveis de estado são derivações de entradas e saídas usando o operador *shift*. No entanto, as formas diretas tornam o controlador sensíveis à erros numéricos (*i.e.*, truncamentos e arredondamentos), que são mais evidentes em implementações de ponto-fixo.

Neste trabalho, três diferentes formas diretas de realização são utilizadas: a Forma Direta I, Forma Direta II e Forma Direta II Transposta. Tais formas são tipicamente compostas por multiplicadores, somadores e memórias para realizar a equação de diferença dos controladores, mas com diferentes ordens nas operações numéricas.

As Figuras 2.1 e 2.2 mostram respectivamente, a representação da Forma Direta I e sua implementação equivalente em ANSI-C, onde os ganhos a_i e b_i são os coeficientes de z^{-1} retirados da Equação (2.2). Nessa representação, pode-se observar dois subsistemas conectados em formato cascata, $h1(z)$ e $h2(z)$.

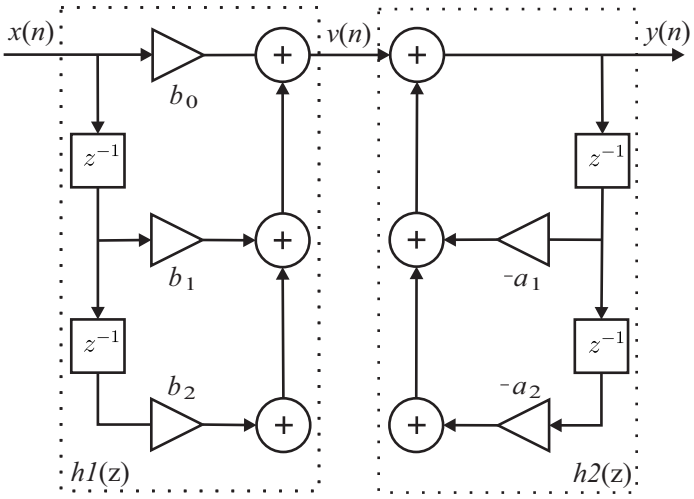

Figura 2.1: Realização na Forma Direta I.

```
1  void direct_form_1(double x[], int x_size, double b[], int b_size, double a
       [], int a_size, double y[]){
2      int i = 0; int j = 0;
3      /* system 1 h1(z) */
4      double v[x_size];
5      for(i = 0; i < x_size; i++){
6          v[i] = 0;
7          for(j = 0; j < b_size; j++){
8              if (j > i) break;
9                  v[i] = v[i] + x[i-j] * b[j];
10             }
11         }
12     /* system 2 h2(z) */
13     y[0] = v[0];
14     for(i = 1; i < x_size; i++){
15         y[i] = 0;
16         y[i] = y[i] + v[i];
17         for(j = 1; j < a_size; j++){
18           if (j > i) break;
19           y[i] = y[i] + y[i-j] * ((-1) * a[j]);
20         }
21     }
22 }
```

Figura 2.2: Exemplo de código em ANSI-C para a realização na Forma Direta I.

As Formas Direta II e Direta II Transposta são chamadas de representações canônicas, pois possuem o menor número possível de memórias (*i.e.*, sem memórias redundantes) [44], como pode ser visto na Figura 2.3. Além disso, ambas diferem entre si apenas no fluxo do sinal.

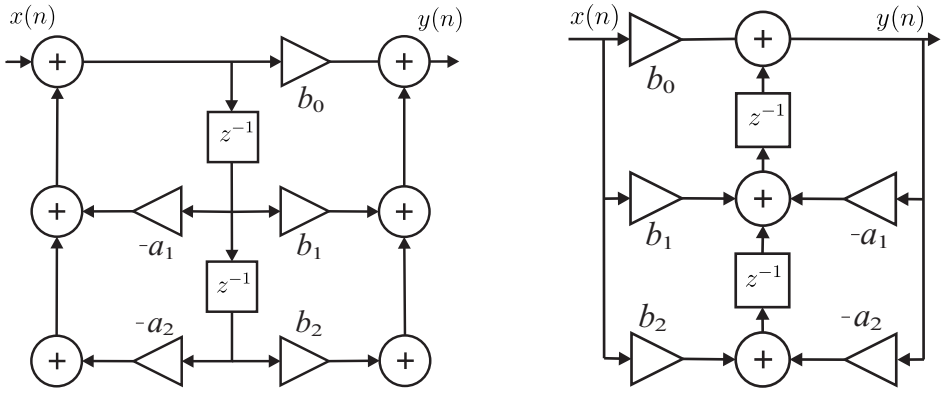

Figura 2.3: Realização nas Formas Direta II e Direta II Transposta.

2.2.2 Formas Delta

Middleton e Goodwin propuseram uma alternativa para minimizar os efeitos de palavra finita (*Finite Word Length*, FWL) em controladores digitais implementados em ponto-fixo, usando o operador *delta* (δ), definido por:

$$\delta = \frac{q-1}{\Delta}, \tag{2.3}$$

onde q é um operador de deslocamento (*e.g.*, o z^{-1}), Δ é o coeficiente de otimização para este domínio, por exemplo, o tempo de amostragem, e δ é a aproximação de Euler para a derivada [45].

Um sistema discreto representado pela Equação (2.1) no domínio z apresenta coeficientes diferentes no domínio δ, sendo descrito pela seguinte função de transferência:

$$H(\delta) = \frac{\bar{b_0} + \bar{b_1}\delta^{-1} + ... + \bar{b_M}\delta^{-M}}{1 + \bar{a_1}\delta^{-1} + ... + \bar{a_N}\delta^{-N}}, \tag{2.4}$$

onde cada coeficiente $\bar{a_i}$ com $0 \leq i \leq N$ e $\bar{b_j}$ com $1 \leq j \leq M$, pode ser descrito por:

$$\bar{a_i} = \binom{k+i}{i} \sum_{k=i}^{N} a_k \Delta^i, \qquad \bar{b_j} = \binom{k+j}{j} \sum_{k=j}^{M} b_k \Delta^j. \tag{2.5}$$

Além disso, um sistema na forma delta tem exatamente o mesmo comportamento de um sistema contínuo amostrado [45], apresentando melhor desempenho sobre arredondamentos e melhorando a precisão dos coeficientes na representação de ponto-fixo, tornando o controlador menos sensível à erros numéricos.

A Figura 2.4 mostra a região de convergência do domínio z e do domínio δ. Enquanto a região de convergência do domínio z é caracterizada pelo circulo unitário, no domínio delta, a região varia conforme o coeficiente de otimização Δ. A medida que Δ tende a zero, o domínio de estabilidade discreto fica mais próximo do domínio contínuo.

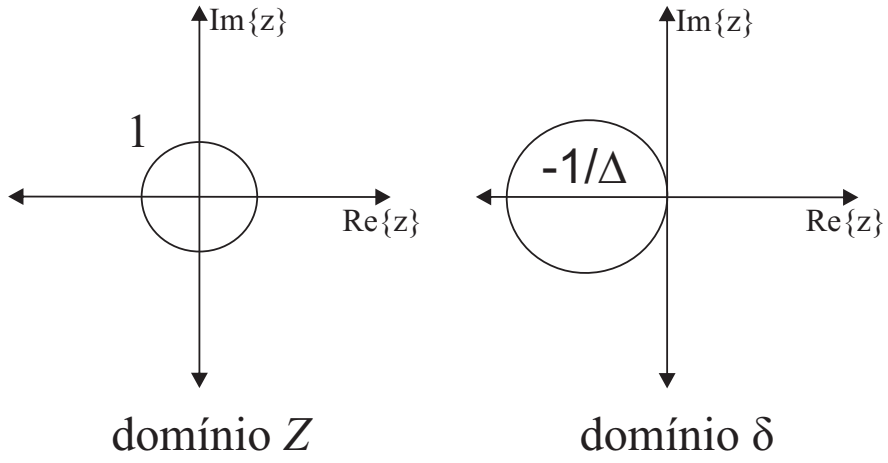

Figura 2.4: Região de convergência do domínio Z e do domínio δ.

O operador *delta* é equivalente ao operador *shift* e seus coeficientes podem ser embarcados nas realizações das formas diretas, de modo que, toda a análise que é feita para o operador *shift* pode ser aplicada ao operador *delta*.

2.3 Representação em Ponto-Fixo

As implementações dos controladores digitais em processadores de ponto-fixo estão sujeitas aos efeitos da palavra finita, as quais estão relacionados às pequenas imprecisões e arredondamentos indesejados que podem levar a problemas funcionais (*e.g.*, instabilidade). Um fato importante é que os arredondamento são feitos durante as quantizações.

Tais quantizações ocorrem durante o processo de conversão de analógico para digital, o qual consiste na aproximação dos valores analógicos para valores discretos. Esse processo gera erros de arredondamento devido ao máximo valor representável, descrito por 2^{-b-1}, onde b é o número de *bits* da parte fracionária.

Um modelo realista considerando palavra finita deve incluir a quantização de todos os valores numéricos, incluindo cada resultado aritmético (*e.g.*, somas e produtos), entrada de sinais e coeficientes. Esses erros de quantização acumulados podem afetar a posição dos polos

e zeros do controlador digital, principalmente nas formas diretas, que faz o controlador perder suas características de estabilidade e fase mínima. Na literatura isso é chamado de fragilidade de controladores [46].

Na fase de projeto, engenheiros de controle evitam posicionar os polos e zeros em regiões que podem ser fatalmente afetadas pelos efeitos FWL (*i.e.*, próximos à borda do circulo unitário). No entanto, existem situações em que esse posicionamento não é feito facilmente, ou não é desejado no contexto. Como por exemplo, nos controladores ressonantes em que polos e zeros são posicionados próximo às bordas de estabilidade. Os efeitos da aritmética de ponto-fixo sobre os controladores ressonantes é estudada por Harnefors [10] e Peretz *et al.* [47].

Uma representação particular em ponto-fixo $< k,l >$, onde k é o número de *bits* da parte inteira e l é o número de *bits* da parte fracionária, representa valores no intervalo de -2^{k-1} até $2^{k-1}-2^{-l}$. Como por exemplo, a representação $< 3,4 >$ (3 *bits* na parte inteira e 4 *bits* na parte fracionária), é capaz de representar valores no intervalo de -4.0 até 3.9375. De modo que, resultados de somas ou multiplicações que excedam a representação são considerados estouros aritméticos.

Um microprocessador geralmente trata um estouro aritmético via *wrap-around* (*i.e.*, permite que os valores que ultrapassem o máximo representável, se tornem valores mínimos, seguindo sua proporção) ou saturação (*i.e.*, trava no máximo valor possível para a representação). Suas respectivas representações podem ser vistas na Figura 2.5.

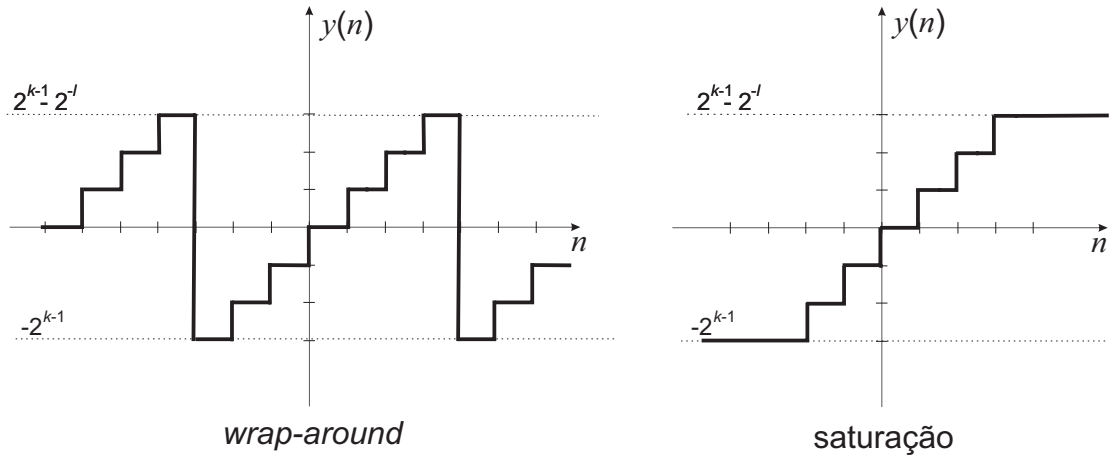

Figura 2.5: Tratamento de estouros aritméticos via *wrap-around* e saturação.

Arredondamentos e estouros aritméticos podem gerar oscilações periódicas, chamadas de ciclo limite. Existem autores que explicam completamente a teoria e operação em ponto-fixo

como em Granas e Dugundji [48]. Adicionalmente, problemas relacionados aos efeitos de FWL em ponto-fixo podem ser amplamente encontrados na literatura [43, 44, 46, 49]. Desta forma, este trabalho propõe uma nova metodologia para verificação de modelos de controladores digitais, considerando os efeitos da palavra finita.

2.3.1 Efeitos de Arredondamentos e Sensibilidade dos Polos e Zeros

Os problemas relacionados à precisão dos controladores digitais são bem conhecidos [6, 8, 9, 46, 50]. Existem essencialmente duas alternativas para tal tratamento: a primeira é simples porém custosa, consiste no aumento do tamanho da palavra (*i.e.*, utilizar um processador com mais *bits*), contudo é necessário um *hardware* mais caro, ocasionando um aumento no custo do produto final; a segunda consiste na otimização e minimização destes efeitos por meio de testes e simulações.

Alguns autores apresentam uma visão geral sobre as principais técnicas para minimizar a degradação de desempenho nos filtros e controladores digitais, causados pelos efeitos de quantização e arredondamentos, e como alcançar uma realização ótima [2, 46, 51].

Os efeitos de arredondamento são especialmente danosos quando os coeficientes são afetados. Estes são responsáveis pela disposição dos polos e zeros e, consequentemente, mudam o comportamento desejado do sistema.

Existem várias ferramentas e técnicas que tratam modelos com incertezas e perturbações (*i.e.*, controle H_∞), produzindo controladores ótimos e robustos. No entanto, Keel e Bhattacharyya mostraram que esses controladores ainda apresentam propriedades indesejadas (*e.g.*, instabilidade) [52]. Além disso, técnicas como controle não frágil também são propostas para minimizar a fragilidade em controladores digitais [2].

2.3.2 Estouro Aritmético

O estouro aritmético é um problema conhecido em controladores digitais. Tal problema pode ocorrer em qualquer nó da realização do controlador digital e é ocasionado por valores que ultrapassam o intervalo disponível pela palavra finita (*i.e.*, quantidade de *bits* na parte inteira), resultando em não linearidades na saída. Em geral, esses eventos podem ser evitados.

Jackson *et al.* [53] mostram que se o resultado computacional do controlador digital, implementado em aritmética de complemento de dois, não excede o intervalo numérico da

palavra finita, então os passos intermediários da computação podem ter algumas ocorrências de estouros aritméticos sem causar acúmulo de erros.

Uma condição necessária e suficiente para evitar estouros aritméticos para qualquer sinal de entrada é garantir que a saída será limitada a v_m, onde v_m corresponde ao limite do intervalo da palavra finita definida no processador.

Neste trabalho, afim de evitar estouros aritméticos durante a realização do controlador digital, a escolha do número de *bits* da parte inteira será baseada no método clássico proposto por Carletta *et al.* [12]. A metodologia utiliza como base o somatório da resposta ao impulso, e é dada por:

$$k = \left\lceil \log_2 \left(\|g_{v_i}, u\| . \|u\|_\infty \right) \right\rceil + 1, \tag{2.6}$$

onde $\|g_{v_i}, u\|$ é o somatório da resposta ao impulso do sistema digital, definido por $\|g_{v_i}, u\| = \sum_{k=0}^{\infty} |h(k)|$ e $\|u\|_\infty$ é o máximo valor do intervalo dinâmico de entradas, definidos pelo conversor analógico digital.

Dependendo das características do controlador digital, o método de Carletta *et al.* pode sugerir um número elevado de *bits* não disponível na plataforma de *hardware* onde o mesmo será embarcado. Afim de diminuir o número de *bits* necessários na parte inteira, pode-se utilizar o escalamento dos sinais de entrada (*scaling*), definido pelo Lema 1.

Lema 1 *Considere um controlador digital com saída $y(n)$ e entrada $x(n)$, limitada em magnitude por v_m, para todo n, multiplicado pelo fator de escala λ. A saída $y(n)$ será limitada por v_m para todo n e para todos os λ que são:*

$$\lambda \leq \frac{1}{\|h(n)\|}, \tag{2.7}$$

onde $h(n)$ é a resposta ao impulso do controlador digital, e $\|\bullet\|$ é definido por: $\|u(n)\| = \sum_{k=0}^{\infty} |u(k)|$.

De modo geral, o escalamento de sinais deve ser utilizado com moderação, visto que o fator de escala impacta diretamente na razão sinal-ruído [44]. Além destes métodos, outros estudos também apresentam técnicas para otimizar o número de *bits* utilizados no formato de ponto-fixo [13, 54–61] ou para minimizar efeitos de arredondamento [44, 62, 63].

2.3.3 Oscilações de Ciclo Limite

As oscilações de ciclo limite (*Limit Cycle Oscilations*, LCO) em controladores digitais são definidos pela presença de oscilações na saída [46]. Essas oscilações podem ser danosas para o sistema de controle, degradando as ações do controlador, causando danos à planta física (em especial, para sistemas mecânicos).

Alguns autores têm estudado a consequência da presença de ciclo limite em sistemas digitais. Peterchev e Sanders mostram que a presença de LCO em conversores de modulação por largura de pulso (*Pulse-Width Modulation*, PWM) pode aumentar o desperdício de energia e diminuir o tempo de vida dos dispositivos [64]. O mesmo problema também foi estudado para controladores ressonantes por Peretz e Ben-Yaakov [47].

LCOs são tipicamente classificados como granulares ou ciclo limite provenientes de estouros aritméticos. Os granulares são oscilações autônomas originárias pela quantização em *bits* menos significativos [44]. As oscilações de ciclo limite provenientes de estouros aritméticos acontecem quando estes eventos são tratados via *wrap-around*. Para este caso, ausência de LCO pode ser assegurada se estouros aritméticos forem tratados com saturação.

Os primeiros estudos que utilizavam o uso de busca exaustiva para garantir a ausência de LCOs em implementações de filtros digitais e controladores surgiram em 1990 [65–69], e desde então pesquisas relacionadas são utilizadas, tais como otimização do espaço de busca aplicando limites e heurísticas [66, 70, 71]. Recentemente, algoritmos de Monte Carlo estão sendo utilizados para detectar LCOs [26].

2.4 Sistemas de Controle em Malha Fechada

Sistemas de controle em malha fechada, também conhecidos como sistemas de controle com realimentação, são caracterizados por manter uma relação pré-estabelecida entre a saída produzida e um determinado valor desejado de referência [72].

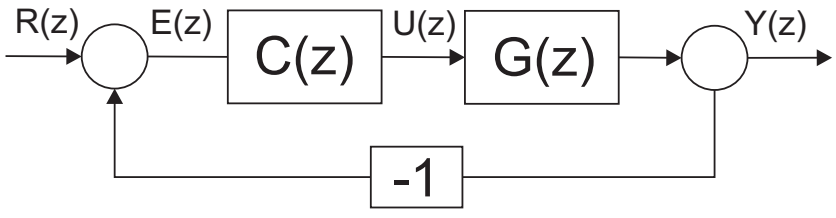

Figura 2.6: Sistema de controle digital em malha fechada.

A Figura 2.6 mostra o diagrama básico de um sistema de controle em malha fechada composto por um controlador $C(z)$ e uma planta $G(z)$. O erro atuante $E(z)$, formado pela diferença entre a saída $Y(z)$ e um sinal desejado de referência $R(z)$ é utilizado como entrada no controlador. O controlador produz um sinal de controle $U(z)$ que reduz o erro e mantém a saída estável.

No entanto, em sistemas com realimentação, é importante levar em consideração algumas incertezas. Geralmente, plantas representadas pela função de transferência $G(z)$ são baseadas em modelos lineares. É natural que o projeto de controle considere incertezas, sejam elas variações paramétricas, provenientes de mudanças em condições ambientais, envelhecimento, dinâmicas não modeladas e não linearidades. Outro fator que também deve ser levado em consideração é a presença de perturbações, que são caracterizadas por sinais indesejados que afetam a saída do sistema, e ainda ruídos, que podem aparecer nas medições de saída. A Figura 2.7 mostra um sistema de controle em malha fechada mais completo, considerando a presença de perturbações $D(z)$ e ruídos $W(z)$.

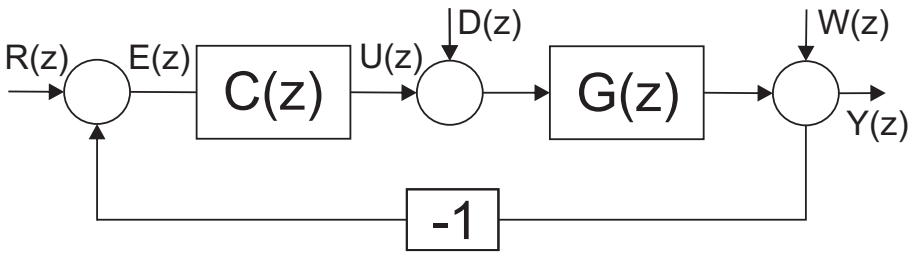

Figura 2.7: Sistema de controle digital com perturbações e ruído.

Entre as diferentes estratégias para representar incertezas, nesse trabalho será adotada uma representação usual, de tal forma que a função de transferência da planta $G(z)$ da Figura 2.7 pode ser representada incluindo perturbações e incertezas paramétricas por $G_p(z)$, definido na Equação (2.8), onde $\Delta G(z)$ é o limite adicional de incertezas.

$$G_p(z) = G(z) + \Delta G(z) \tag{2.8}$$

2.4.1 Estabilidade

Considere um sistema linear de uma única entrada e uma única saída (*Single-Input Single-Output*, SISO), discreto e invariante no tempo (*Linear Time-Invariant*, LTI), representado por:

$$G(z) = \frac{b_0 + b_1 z^{-1} + ... + b_M z^{-M}}{1 + a_1 z^{-1} + ... + a_N z^{-N}}. \tag{2.9}$$

Usualmente, um sistema linear no domínio discreto é dito assintoticamente estável se todos os seus polos da função de transferência estão dentro do círculo unitário (sem cancelamento dos polos e zeros). Além disso, tal sistema apresenta entradas e saídas limitadas (*Bounded-Input Bounded-Output*, BIBO) se, e somente se, todos os polos encontram-se dentro do círculo unitário.

Nesta seção, serão seguidas ideias semelhantes as citadas por Fadali [41] para introduzir a noção de variação interna de estabilidade, além da própria função de transferência em um sistema em malha fechada.

Considerando a configuração padrão descrita pela Figura 2.7 e a escolha das saídas $Y(z)$, entradas $U(z)$, uma referência $R(z)$, uma entrada de perturbação $D(z)$ e uma medida de ruído $W(z)$, temos:

$$\begin{bmatrix} Y \\ U \end{bmatrix} = \begin{bmatrix} \frac{G(z)C(z)}{1+G(z)C(z)} & \frac{G(z)}{1+G(z)C(z)} & \frac{1}{1+G(z)C(z)} \\ \frac{C(z)}{1+G(z)C(z)} & \frac{-G(z)C(z)}{1+G(z)C(z)} & \frac{-C(z)}{1+G(z)C(z)} \end{bmatrix} \begin{bmatrix} R \\ D \\ W \end{bmatrix}. \tag{2.10}$$

Definição 1 – *Se toda relação entre função de transferência e as entradas do sistema são BIBO estáveis, então o sistema é chamado de internamente estável [41].*

Teorema 1 – *O sistema descrito pela Figura 2.7 é internamente estável se, e somente se, todos os polos do sistema em malha fechada estão dentro do circulo unitário [41].*

Pode-se concluir a partir do Teorema 1 (cujo a prova é dada em [41]), que a relação entre um controlador $C(z) = N_C(z)/D_C(z)$ e sua planta $G(z) = N_G(z)/D_G(z)$, com característica polinomial

$$S(z) = N_C(z)N_G(z) + D_C(z)D_G(z), \tag{2.11}$$

é internamente estável se os zeros estão dentro do circulo unitário. Para isso, precisamos considerar que o controlador $C(z)$ é assintoticamente estável.

Neste trabalho, utilizaremos a verificação de estabilidade baseada no critério de Jury, explicado na Seção 3.1.1, para avaliar a Equação (2.11).

2.5 Teorias do Módulo da Satisfatibilidade (SMT)

SMT (*Satisfiability Module Theories*) verifica a satisfatibilidade de fórmulas de primeira ordem a partir de uma ou mais teorias de fundamentação, que são compostas por um conjunto de sentenças. De modo formal, $\Sigma - theory$ é uma coleção de sentenças sobre a assinatura Σ. Dada uma teoria T, nós dizemos que φ é um modulo satisfatível de T se $T \cup \{ \varphi \}$. Em outra definição, podemos dizer que uma teoria T é definida como uma classe de estruturas e φ é um módulo satisfatível se existe uma estrutura M em T que satisfaz φ (*i.e.*, $M \models \varphi$) [73].

Solucionadores SMT como Z3 [74] e Boolector [75] suportam diferentes tipos de teorias, de modo que o seu desempenho pode variar conforme a sua implementação.

Teoria	Exemplo
Igualdade	$x1 = x2 \wedge \neg(x2 = x3) \Rightarrow \neg(x1 = x3)$
Aritmética Linear	$(4y_1 + 3y_2 \geq 4) \vee (y_2 - 3y_3 \leq 3)$
Vetores de *bit*	$(b >> i)\&1 = 1$
Arranjos	$store(a, j, 2) \Rightarrow a[j] = 2$
Teorias Combinadas	$(j \leq k \wedge a[j] = 2) \Rightarrow a[i] < 3$

Tabela 2.1: Exemplos de teorias suportadas.

A Tabela 2.1 mostra algumas das teorias suportadas pelos solucionadores SMT utilizados nesse trabalho. A teoria de igualdade permite verificações de igualdade e desigualdade entre predicados utilizando os operadores $(=)$ $(\leq)$ $(<)$. A teoria da aritmética linear é responsável apenas pelas funções aritméticas (adição, subtração, multiplicação e divisão) entre variáveis e constantes numéricas. A teoria de vetores de *bit* permite operações *bit* a *bit* considerando diferentes arquiteturas (*e.g.*, 32 e 64 *bits*), nela estão presentes os operadores: e (&), ou (|), ou-exclusivo ($\oplus$), complemento ($\sim$), deslocamento para a direita ($\gg$) e deslocamento para a esquerda ($\ll$). Além disso, a teoria de arranjos permite manipulação de operadores como *select* e *store*.

Diferentemente das fórmulas geradas na satisfação booleana, que são apenas compostas por variáveis booleanas, as quais podem assumir valores verdadeiro e falso e conectivos lógicos, as fórmulas de primeira ordem são formadas por conectivos lógicos, variáveis, quantificadores, funções e símbolos de predicado [73]. Em geral, as teorias do módulo da satisfatibilidade tem sido exploradas em diversas aplicações [76], apresentando melhores resultados (se comparado à satisfação booleana), incluindo o suporte a diferentes teorias de decisão [73, 74].

2.6 Verificação de Modelos Limitada

A verificação formal de *software* e sistemas é uma importante tarefa para garantir que um modelo atenda aos seus requisitos. Contudo, para sistemas robustos e complexos, tal procedimento se torna difícil. Como as técnicas de verificação de modelos não requerem provas (métodos algorítmicos ao invés de dedução), elas têm ganhado um lugar importante no paradigma de verificação, uma vez que os sistemas se tornam mais complexos (*e.g.*, ciberfísicos), que demandam curtos ciclos de desenvolvimento e auto nível de confiança [77].

A competição internacional de verificação de software (*International Competition on Software Verification*, SV-COMP) [78], é uma competição anual para as ferramentas de verificação de *software*, que avalia diferentes técnicas de teste e verificação. Uma importante técnica que tem apresentado vários resultados interessantes nos últimos anos, é a verificação limitada de modelos (*Bounded Model Checking*, BMC). A técnica BMC pode ser baseada em satisfação booleana (*Boolean Satisfatibility*, SAT) [79] ou em teorias do módulo da satisfatibilidade (*Satisfatibility Modulo Theories*, SMT) [80]. Ambas têm sido amplamente aplicadas com sucesso para a verificação de programas sequenciais e concorrentes, e também para encontrar problemas em programas reais [81–84]. No entanto, a aplicação destas técnicas para garantir a corretude em sistemas de tempo discreto é recente [36, 37, 39, 40, 85].

A ideia básica da técnica BMC é checar a negação de uma determinada propriedade até uma determinada profundidade. Basicamente, a técnica BMC utiliza sistemas de transição de estados para modelar os possíveis comportamentos do *software*. Tais sistemas, são compostos por grafos, onde os vértices representam os estados (*i.e.*, valor das variáveis e o contador de programa) e as transições descrevem como o sistema se movimenta entre esses estados (*i.e.*, condições) [86].

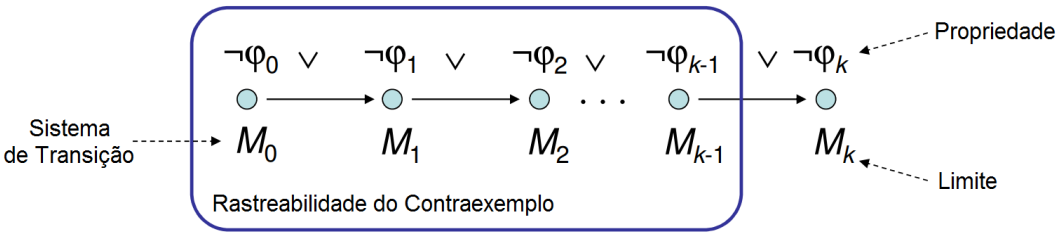

Figura 2.8: Representação gráfica de um sistema de transição de estados.

A Figura 2.8 mostra um sistema transição M, o qual contém uma propriedade ϕ e um limite k. A técnica BMC percorre o sistema até a profundidade especificada e o traduz em

condições de verificação ψ (*Verification Condition*, VC), de modo que ψ é satisfatível, se e somente se, ϕ tem um contraexemplo a uma profundidade menor ou igual a k. Em geral, contraexemplos contêm todos os estados utilizados até a propriedade ser violada [87]. De posse de um contraexemplo, o usuário pode facilmente reproduzir a execução do sistema até o estado de erro.

Em BMC de controladores digitais, o limite k é o número de iterações e chamadas recursivas dentro das realizações dos controladores. Assim, a técnica BMC gera as condições de verificação que refletem exatamente o caminho e estados a serem executados, os contextos em que as funções são chamadas e as expressões relacionadas à precisão de *bits* [81].

2.7 *C Bounded Model Checker*

CBMC (*C Bounded Model Checker*) é uma ferramenta escrita em C++ que aplica a técnica BMC utilizando solucionadores de satisfação booleana, para verificar formalmente programas ANSI-C com extensão C89, C99 e C11. O CBMC suporta diversas funcionalidades da linguagem ANSI-C, incluindo a construção de ponteiros, alocação dinâmica de memória, recursão e diversos tipos numéricos. A ferramenta também possui uma biblioteca interna para operações de aritméticas em ponto-fixo e em ponto-flutuante, e é capaz de checar propriedades como: segurança de ponteiros, limites de arranjos, programas concorrentes e assertivas fornecidas pelo próprio usuário [88].

Afim de encontrar violações em programas ANSI-C, o CBMC reduz o problema de verificação de modelos limitados à equações de vetores de *bits*. A arquitetura da ferramenta e o processo de geração das equações é demonstrada na Figura 2.9.

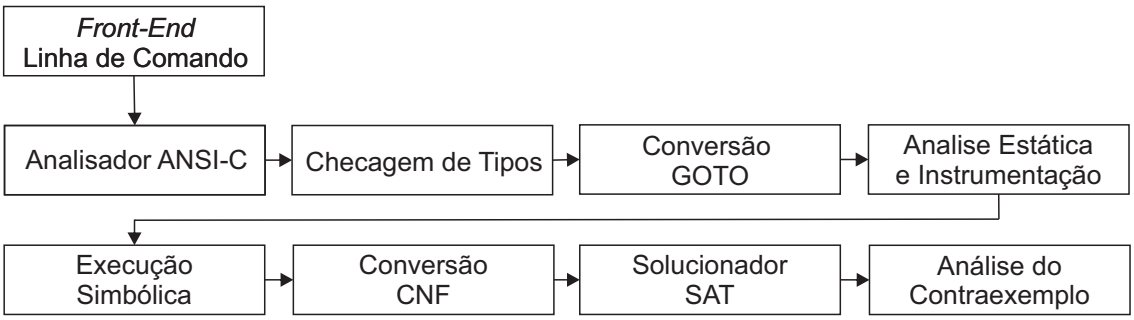

Figura 2.9: Visão geral da arquitetura do verificador CBMC.

Primeiramente, o usuário acessa a ferramenta utilizando o *front-end* via linha de comando do CBMC, informando o código ANSI-C e parâmetros para a verificação (*e.g.*, propriedades a serem verificadas). O código é enviado para um pre-processador da linguagem ANSI-C que extrai uma árvore sintática, contendo informações sobre a sintaxe programa. Após isso, é feita a checagem de tipos e símbolos; caso alguma inconsistência seja detectada (*i.e.*, violação na estrutura da linguagem), a verificação é abortada. O CBMC utiliza a linguagem GOTO como representação intermediária, de modo que, as instruções de controle de fluxo e *loops* da linguagem ANSI-C são traduzidas para seus equivalentes em GOTO. A partir do código GOTO gerado, o verificador de modelos efetua uma análise estática para encontrar problemas de ponteiros inválidos e vazamentos de memória que possam existir no programa. Não identificando nenhum problema, o CBMC efetua uma execução simbólica, considerando o limite de profundade k desejado pelo usuário. Esse procedimento gera equações em SSA (*Single Static Assignment*) que são posteriormente traduzidas em expressões booleanas na forma normal conjuntiva (*Conjunctive Normal Form*, CNF). Por fim, o CBMC gera duas equações: C (para as restrições) e P (para as propriedades), de modo que, a equação $C \wedge \neg P$ é resolvida por um solucionador SAT como MiniSAT [89]. Se a equação é satisfeita, uma violação foi encontrada e um contraexemplo é gerado.

2.8 *Efficient SMT-Based Context Bounded Model Checker*

ESBMC (*Efficient SMT-Based Context-Bounded Model Checker*) é um verificador de modelos capaz de verificar programas C/C++ sequenciais e multitarefa utilizando o *front-end* do CBMC [82, 90, 91]. No entanto, no ESBMC utiliza-se solucionadores SMT em vez de SAT, e suporte para verificação de propriedades expressas em lógica temporal linear [92, 93]. Além disso, o ESBMC tem sido utilizado em diversas aplicações, como por exemplo, na evidência de falhas em partes críticas de códigos escritos em ANSI-C [94], como alternativa no problema de particionamento em *hardware* e *software* [95, 96], na aplicação de indução matemática e invariantes em problemas BMC [97, 98], e ainda na verificação de programas concorrentes em unidades de processamento gráfico [99].

De modo geral, o ESBMC possui apenas uma biblioteca para operações ariméticas em ponto-fixo para verificação de filtros e controladores digitais [38–40, 85]. Além disso, o verificador possui suporte a operações a nível de *bit*, ponteiros, estruturas e uniões, permitindo a veri-

ficação por violações em propriedades relacionadas à segurança de ponteiros, limite de vetores, atomicidade, estouro aritmético, bloqueio fatal, corrida de dados e vazamento de memória, para programas sequenciais e paralelos (com memória compartilhada e *mutex*).

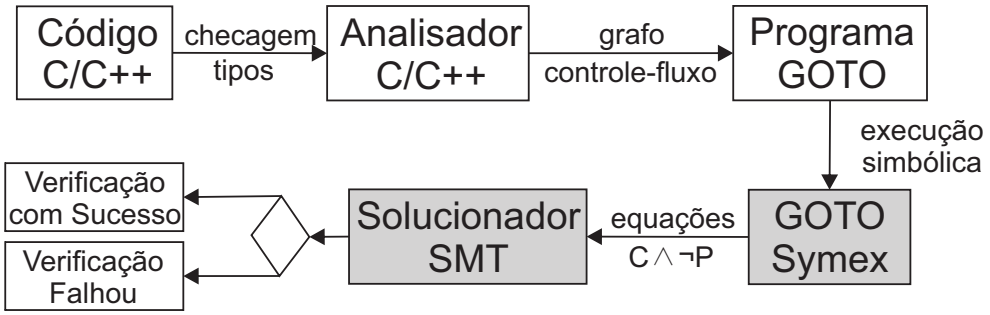

Figura 2.10: Visão geral da arquitetura do verificador ESBMC.

A Figura 2.10 mostra a arquitetura do ESBMC. A verificação utiliza como entrada um código C/C++ fornecido pelo usuário e um conjunto de parâmetros (*e.g.*, limite k, tempo de verificação e propriedades). Primeiramente, o código é enviado para um analisador léxico e sintático capaz de verificar se sua estrutura está coerente com a linguagem desejada. Após a validação, é gerado um grafo de fluxo de controle (*Control-Flow Graph*, CFG) com todos os possíveis estados do algoritmo. A partir do grafo, o programa é reescrito na forma GOTO (*i.e.*, expressões como *switch* e *while* são transformadas em expressões *goto*), e é executado simbolicamente para geração de equações de atribuição estática única (SSA). O ESBMC gera duas equações SMT: C (para as restrições) e P (para as propriedades), de modo que, a equação $C \wedge \neg P$ pode ser resolvida por diversos solucionadores suportados pela ferramenta, dentre eles Z3 [74] e Boolector [75].

Nesse trabalho, o ESBMC foi utilizado como um dos verificadores de modelos limitados, pois representa uma das mais eficientes ferramentas de verificação que utiliza a técnica BMC [100–102]. Em especial, o ESBMC é uma das ferramentas mais eficientes para verificar problemas que envolvem aritmética de vetores de *bit*, de acordo com a edição 2015 do SV-COMP [78].

Dentro do ESBMC, o código ANSI-C associado é formulado pela construção da seguinte fórmula lógica:

$$\psi_k = I(S_0) \wedge \bigvee_{i=0}^{k} \bigwedge_{j=0}^{i-1} \gamma(s_j, s_{j+1}) \wedge \overline{\phi(s_1)}, \tag{2.12}$$

onde ϕ é uma propriedade (*e.g.*, estouro aritmético), S_0 é um conjunto de estados iniciais do sistema de transição M, e $\gamma(s_j, s_{j+1})$ é a relação de transição de M entre os passos de tempo j e $j+1$. Assim, $I(S_0) \wedge \bigwedge_{j=0}^{i-1}$ representa a execução das relações de transição M no comprimento i. A condição de verificação acima ψ pode ser satisfeita se, e somente se, para algum $i \leq k$, existe um estado acessível i no qual ϕ é violado.

Se a Equação (2.12) é satisfeita, então o solucionador SMT fornece uma atribuição de satisfação, onde os valores das variáveis do controlador digital são extraídas para a construção de um contraexemplo.

2.9 Resumo

Neste capítulo, foram introduzidos os conceitos básicos para o entendimento deste trabalho, tais como, definição de controladores digitais, suas formas de representação (em especial a equação de diferença), suas realizações, o domínio delta e suas vantagens, representação em ponto fixo, efeitos da palavra finita, bem como a definição de estouro aritmético, ciclo limite e estabilidade considerando incertezas. Foram também apresentados conceitos sobre verificação formal de *software* utilizando a técnica de modelos limitados (BMC), além dos diferentes verificadores que aplicam a respectiva técnica, dentre eles, CBMC (*C Bounded Model Checker*) e ESBMC (*Efficient SMT-Based Context-Bounded Model Checker*). Por fim, conceitos básicos relacionados com as teorias SMT foram abordados. Como resultado, o conteúdo apresentado nesse capítulo fornece todo o embasamento necessário para compreensão do trabalho desenvolvido, que será descrito nas seções subsequentes.

Capítulo 3

Verificação de Controladores Digitais em Ponto-Fixo

Neste capítulo, será apresentada a metodologia de verificação de controladores digitais, que emprega a técnica BMC baseada em solucionadores SAT e SMT. Em especial, será descrito o procedimento de verificação de controladores digitais para as seguintes propriedades: estouros aritméticos, ciclo limite, restrições temporais, estabilidade e fase mínima. Além disso, a verificação de estabilidade para controladores digitais em malha fechada considerando incertezas paramétricas na planta. Para tais verificações, são utilizadas realizações nas formas diretas e delta.

3.1 Metodologia de Verificação

Este trabalho descreve uma nova metodologia para verificação de controladores digitais implementados em ponto-fixo. Essa metodologia é suportada pelo verificador de sistemas digitais (*Digital-Systems Verifier*, DSVerifier)[1]. Tal ferramenta pode ser conectada ao ESBMC [81] e ao CBMC [83], e sua utilização é mostrada com mais detalhes no Apêndice A.

3.1.1 Verificação de Controladores Digitais

A Figura 3.1 mostra uma visão geral da metodologia proposta para verificar os controladores digitais. No passo 1, o controlador é inicialmente projetado com qualquer técnica

[1]Disponível em: http://dsverifier.org

e ferramenta. Após o projeto, as características de implementação devem ser definidas, como mostradas nos passos 2 e 3: o formato da palavra finita (número de *bits* na parte inteira e na parte fracionária), intervalo dinâmico da entrada e forma de realização (diretas ou delta). No passo 4 são definidas as configurações da verificação (*e.g.*, plataforma de *hardware*, tempo máximo de verificação e limite k). Depois destas configurações, o processo de verificação é iniciado no passo 5 com a escolha da ferramenta de verificação (atualmente são suportados pelo DSVerifier, os verificadores ESBMC [81] e CBMC [83]).

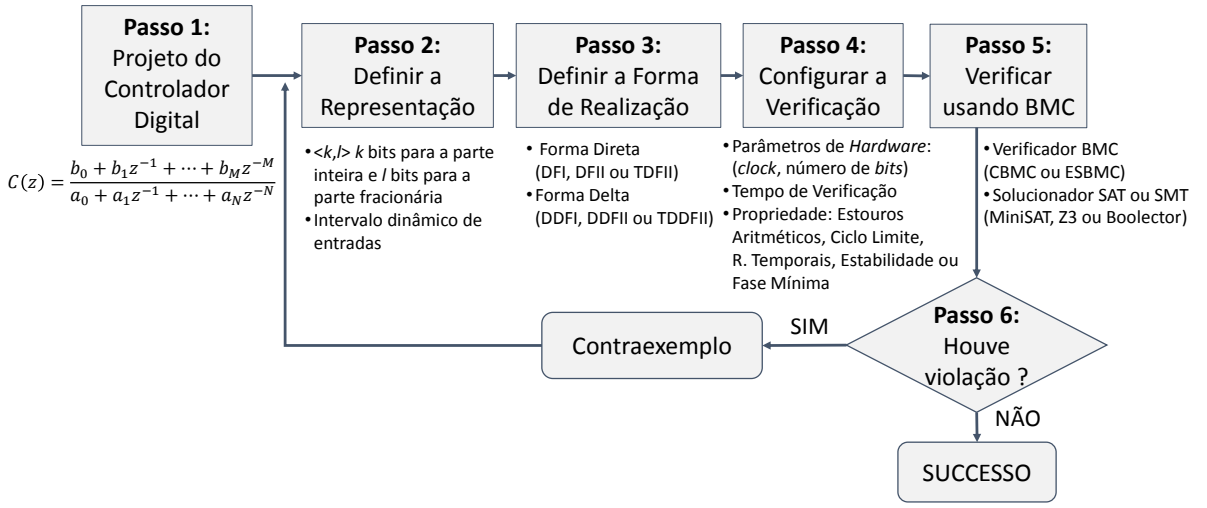

Figura 3.1: Metodologia proposta para a verificação de controladores digitais.

O DSVerifier então checa a propriedade desejada e no passo 6 retorna o resultado da verificação, que pode ser: sucesso (caso não haja nenhuma violação da propriedade para a implementação proposta), ou caso contrário, a ferramenta retorna falha e mostra um contra-exemplo que contém as entradas utilizadas e os estados. A partir deste contraexemplo, outra opção de implementação (*i.e.*, realização ou representação) pode ser escolhida, afim de evitar a falha. Esse procedimento pode ser repetido até que a implementação do controlador digital não apresente falhas na profundidade k.

Verificação de Estouros Aritméticos

Ao utilizar processadores com aritmética de ponto-fixo, deve-se tomar um cuidado especial com estouros aritméticos, que são difíceis de serem detectados sem ferramentas computacionais e geralmente acontecem em tempo de execução, durante o processo de quantização.

A verificação é configurada para utilizar como entrada valores não determinísticos dentro de um intervalo especificado pelo usuário, podendo assumir todos os possíveis valores dentro deste intervalo (*e.g.,* considerando uma precisão de 2 *bits* e o intervalo -1.0 a 1.0, são consideradas entradas: -1.0, $-0,75$, -0.5, -0.25, 0, 0.25, 0.5, 0.75, 1.0).

Nesse trabalho, assertivas são incluídas para cada saída produzida pelo controlador digital. Na prática, assertivas são funções que avaliam as condições lógicas. Caso uma determinada condição não seja satisfeita, o verificador de modelos é informado e indica um erro. Então, se é produzida uma saída que ultrapassa o limite da representação, a mesma é considerada uma violação de estouro aritmético (seja positivo ou negativo). Deste modo, eventuais estouros aritméticos que ocorram durante os nós intermediários da realização do controlador digital, são desconsiderados com base em Jackson *et al.* [53].

Como consequência, um literal $l_{overflow}$ é definido no DSVerifier com o objetivo de representar a validade de cada saída gerada, de acordo com a seguinte restrição:

$$l_{overflow} \Leftrightarrow (FP \geq MIN) \wedge (FP \leq MAX), \tag{3.1}$$

onde *FP* é a aproximação em ponto-fixo para a saída produzida, e *MIN* e *MAX* são respectivamente os valores mínimos e máximos representados pelo formato em ponto-fixo. Portanto, na verificação de estouros aritméticos, nunca pode acontecer de uma expressão em ponto-fixo ser avaliada com um valor que não pertence ao intervalo da palavra finita.

Um exemplo de estouro aritmético pode ser visto no seguinte controlador digital (retirado dos nossos casos de teste, ver Apêndice B):

$$C_1(z) = \frac{0.10z^3 - 0.28z^2 + 0.26z - 0.08}{z^3 - 2.57z^2 + 2.18z - 0.60}. \tag{3.2}$$

A Figura 3.2 mostra a ocorrência de um estouro aritmético para o controlador descrito pela Equação (3.2) implementado em qualquer forma direta com $< 4,4 >$ (*i.e.*, 4 *bits* para a parte inteira e 4 *bits* para a parte fracionária) e o intervalo de entradas limitado a $[-5.0, 5.0]$. O controlador apresenta um estouro aritmético na sexta saída, com valor 8.7500 (considerando a profundidade de verificação $k = 10$), caso os valores de entrada sejam respectivamente: -3.9375, -0.5625, -5.0000, -0.3750, 3.7500 e -0.4375.

O número de *bits* da parte inteira para esse controlador digital é escolhido baseado em um método conservador descrito por Carletta *et al.* [12]. O somatório do módulo da resposta ao impulso $\|g_{v_i}, u\| = \sum_{k=-\infty}^{\infty} |h_k|$ para esse controlador digital é dado por:

$$\sum_{k=-\infty}^{\infty} |h_k| = 1.199978. \tag{3.3}$$

Considerando que o maior valor utilizado na entrada do controlador é $\|u\|_\infty = 5.0$, Carletta *et al.* demonstram que o número de *bits* da parte inteira deve ser:

$$k = \left\lceil \log_2 \left(\|g_{v_i}, u\| . \|u\|_\infty \right) \right\rceil + 1 = 4. \tag{3.4}$$

Note que o número de *bits* para a parte inteira escolhido para esse controlador é capaz de representar apenas valores no intervalo: -8.0 a 7.9375.

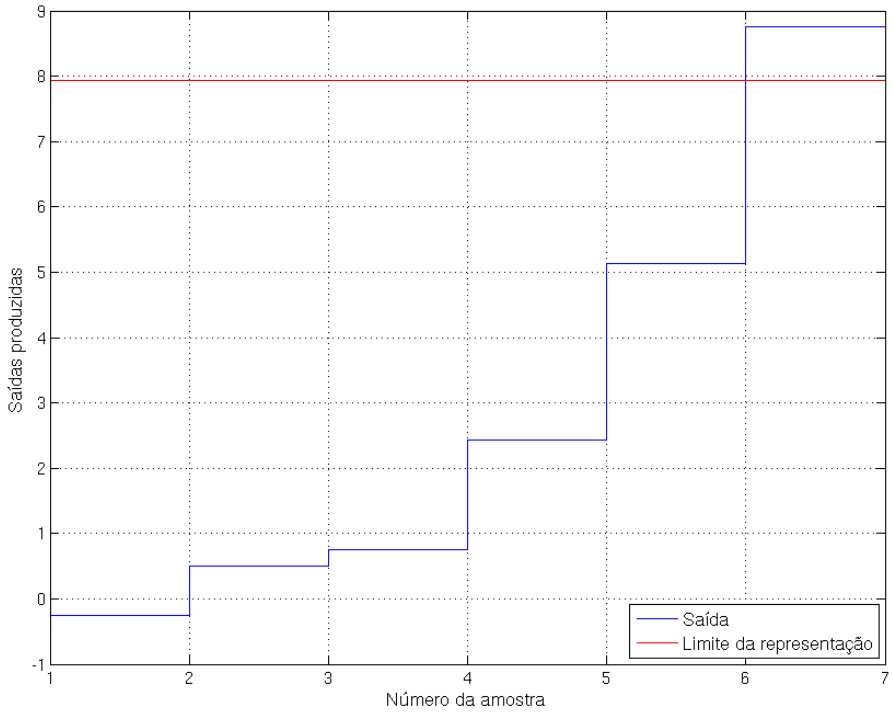

Figura 3.2: Exemplo de estouro aritmético para o controlador digital representado pela Equação (3.2).

Neste contexto, podemos evidenciar a aplicação da técnica BMC para detectar estouros aritméticos em controladores digitais, que não são detectados em métodos conservadores, a qual representa uma importante contribuição para a área de sistemas de controle e verificação formal.

Verificação de Ciclo Limite

Para a verificação da presença de ciclo limite em uma realização de um controlador em ponto-fixo, o bloco de quantização é configurado para permitir *wrap-around* durante os estouros aritméticos que ocorram nas saídas, o que significa que a máquina de verificação não detectará as falhas de estouro aritmético como no caso anterior. Adicionalmente, para essa verificação, são adotados estados iniciais não determinísticos (nos valores das memórias do controlador). O controlador pode ser configurado para receber uma sequência constante de entradas nulas, ou ainda, uma sequência constante de números não determinísticos dentro de um intervalo definido pelo usuário. O controlador é desdobrado para um determinado número de entradas e assertivas são adicionadas para detectar possíveis falhas (se um conjunto de saídas se repete dado uma sequência constante de entradas).

$$\begin{aligned} l_{LCO} \Longleftrightarrow \exists n,k \in \mathbb{N}, \exists c\mathbb{R} | x_m = c \Longrightarrow \exists y_{k+i} = y_{k+n+i}, \\ \forall i \in \{0,1,2,...,n\}, m \in \{k,k+1,k+2,...,k+2n\}. \end{aligned} \tag{3.5}$$

A ocorrência do ciclo limite é definida no DSVerifier pelo literal l_{LCO}, descrito pela Equação (3.5).

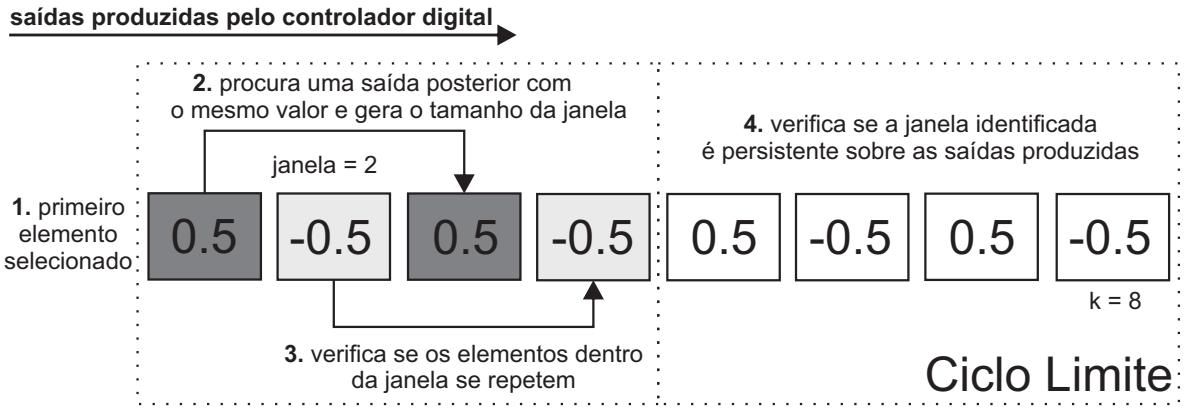

Figura 3.3: Representação gráfica do algoritmo para detectar a presença de oscilações de ciclo limite em controladores digitais.

A Figura 3.3 mostra a representação gráfica do mecanismo para detecção de oscilações de ciclo limite. A detecção é baseada nas saídas produzidas pelo controlador digital. Inicialmente é escolhida a primeira saída como referência, em seguida procura-se nos elementos posteriores um valor igual, afim de encontrar o tamanho da janela de tempo do ciclo limite. A partir de uma janela encontrada, é verificado se os elementos de dentro da janela de tempo se

repetem, caso isso ocorra e haja uma persistência da sequência sobre as saídas, é caracterizada a presença do ciclo limite. Em geral, este método para verificação de ciclo limite difere-se de Abreu *et al.* [39] e Cox [36, 37] por utilizar entradas não determinísticas ao invés de entradas nulas, e também por analisar todas as saídas produzidas (em uma profundidade k), ao invés das memórias do sistema digital. Evitando que oscilações não persistentes sejam consideradas de ciclo limite.

Um exemplo de falha de ciclo limite com entradas nulas é mostrada na Figura 3.4. Caso o controlador digital descrito pela Equação (3.6) (retirado dos nossos casos de teste, ver Apêndice B) seja implementado utilizando $< 11,4 >$ (11 *bits* na parte inteira e 4 *bits* na parte fracionária), com realização na Forma Direta I, e seus estados iniciais sejam $y(-1) = -0.0625$ e $y(-2) = -0.3125$ (onde $y(-1)$ e $y(-2)$ são as memórias do controlador). As saídas do controlador 3.6 oscilarão entre 0.125 e -0.125.

$$C_2(z) = \frac{1.611z^2 + 3.079z - 3.794}{z^2 + 1.084z - 0.1289} \tag{3.6}$$

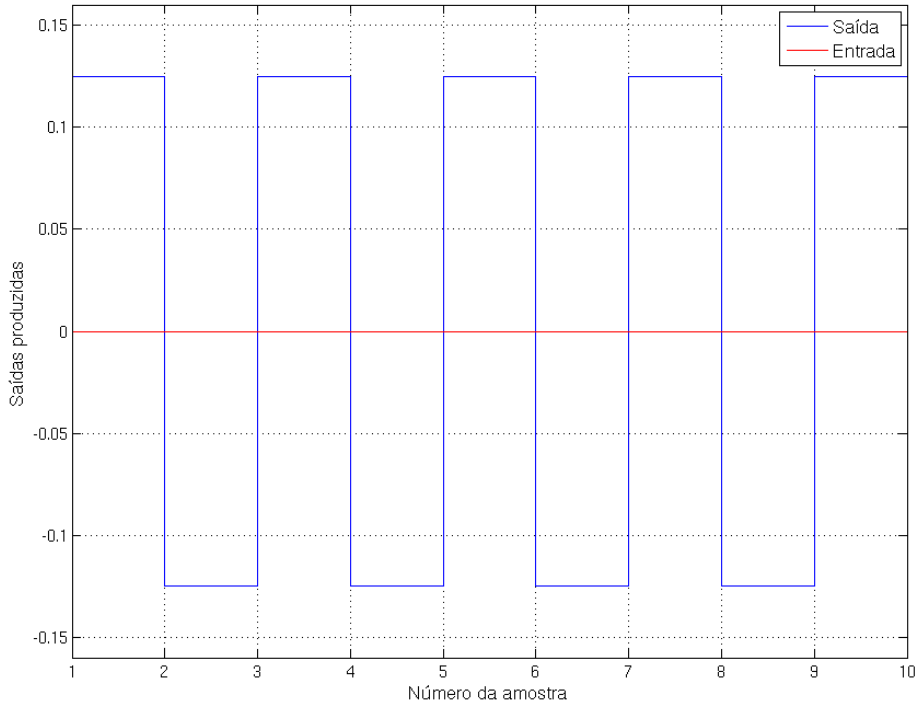

Figura 3.4: Presença de ciclo limite para entradas nulas no controlador digital representado pela Equação (3.6).

Outro exemplo de falha de ciclo limite, desta vez considerando uma sequência não nula de entradas, é mostrada na Figura 3.5. Caso o controlador digital descrito pela Equação (3.7)

(retirado dos nossos casos de teste, ver Apêndice B) seja implementado utilizando $< 4,8 >$ (4 *bits* na parte inteira e 8 *bits* para a parte fracionária), com realização na Forma Direta II e seu estado inicial seja $v(-1) = -0.33984375$ (onde $v(-1)$ é a memória do controlador). Dada uma sequência constante de entradas em -0.01171875, as saídas do controlador oscilarão entre 0.01171875, 0.015625, 0.015625, 0.01953125 e 0.015625.

$$C_3(z) = \frac{1.5610z - 1.485}{z - 0.96} \tag{3.7}$$

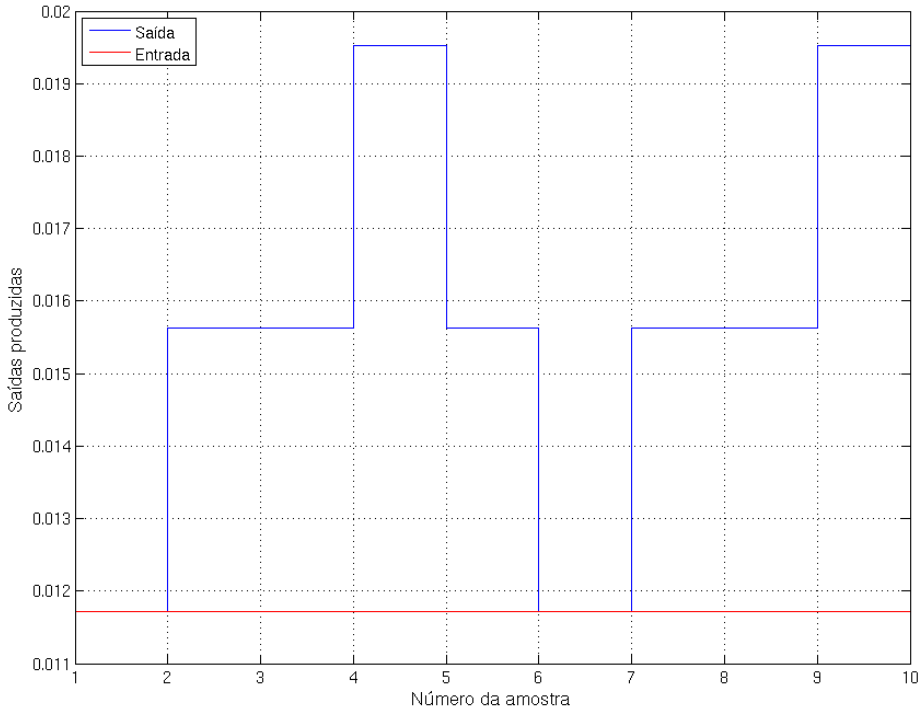

Figura 3.5: Presença de ciclo limite para uma sequência constante não nula no controlador digital representado pela Equação (3.7).

Verificação de Restrições Temporais

Em sistemas de controle digital, o tempo de amostragem é um parâmetro importante e deve ser escolhido cuidadosamente. Visto que, todo o sistema dinâmico é alterado com uma modificação no tempo de amostragem [103].

Uma seleção precisa do período de amostragem é essencial para sistemas controlados por computador. De um lado, curtos tempos de amostragem requerem um maior desempenho e consequentemente processadores com maior frequências de relógio (isso pode impor uma

limitação técnica no projeto de um controlador digital). Por outro lado, tempos de amostragem longos não permitem a reconstrução contínua do sinal [104].

Em princípio, a escolha do tempo de amostragem depende da planta física onde o sistema de controle é aplicado. A escolha correta da implementação computacional do controlador pode reduzir o número de operações aritméticas e consequentemente o custo computacional.

Como um controlador digital é tipicamente um sistema de tempo real, ele não pode levar mais tempo que o tempo de amostragem para processar uma tarefa [105]. Em aplicações práticas, o controlador é projetado com um razoável período de amostragem para produzir bons resultados de simulação. Após isso, é implementado em um sistema de computador, onde as tarefas são escalonadas a cada período de amostragem (*i.e.*, o tempo máximo que o processador leva para realizar todas as tarefas de controle correspondentes a sua operação). Se uma operação não termina no tempo especificado, então a saída produzida pode não estar correta, afetando o funcionamento esperado do sistema de controle.

Por essa razão, a verificação de restrições temporais se torna fundamental em um ferramenta de projeto de controladores, porquê a mesma pode indicar se o período de amostragem e realização computacional são compatíveis antes de efetuar a implementação física, evitando assim sérios problemas relacionados ao funcionamento do sistema.

O tempo necessário para executar um código específico pode ser estimado utilizando o tempo de execução no pior caso (*Worst-Case Execution Time*, WCET) [106]. A Figura 3.6 mostra uma linha de código em ANSI-C utilizada na realização da Forma Direta I de um controlador digital.

```
y[i] = y[i] + y[i-j] * ((-1) * a[j]);
```

Figura 3.6: Exemplo de instrução em ANSI-C utilizada na realização de um controlador digital na Forma Direta I.

A instrução acima pode ser quebrada em um conjunto de instruções em linguagem *assembly*, como mostrado na Figura 3.7. Em particular, cada processador tem uma tabela de ciclos de relógio consumidos para cada instrução *assembly*. Para saber o tempo total necessário para executar um código, o número de ciclos de relógio deve ser divido pela taxa do processador (ou multiplicado pelo tempo de relógio). No entanto, estimar os ciclos de relógio é um desafio, uma vez que a implementação de controladores contém estruturas de repetição e tomadas de

decisão. O controlador pode assim precisar de diferentes ciclos de relógio para sua execução, dependendo dos parâmetros de entrada (que geralmente são valores não determinísticos).

```
1  rol r25
2  in r18,__SP_L__
3  sub r18,r24
4  sbc r19,r25
5  in __tmp_reg__,__SREG__
6  cli
7  out __SP_L__,r18
8  in r25,__SP_H__
9  adiw r24,1
10 std Y+8,r25
```

Figura 3.7: Conjunto reduzido de instruções *assembly* retirados da Figura 3.6.

Para que a metodologia seja capaz de encontrar violações de restrições temporais, é necessário que usuário informe dados relacionados a sua plataforma de *hardware*, como o *clock* e a quantidade de ciclos necessários para realizar determinadas intruções *assembly* utilizadas durante a realização do controlador digital (*e.g.*, *ADD*, *MOV*, *POP*, *PUSH*). Como resultado, um literal l_{timing} é definido no DSVerifier para representar o tempo de resposta, seguindo a restrição:

$$l_{timing} \iff ((N \times T) \leq D), \tag{3.8}$$

onde N é o número de ciclos de relógio gastos pelo controlador digital, T é o período de relógio e D é o tempo máximo para execução das tarefas. Assim, a máquina de verificação checa a satisfatibilidade da Equação (3.8) para garantir o máximo tempo aceitável para as realizações do controlador. De modo geral, esta verificação é apenas uma estimativa, pois não é considerado o escalonamento real das instruções, nem o *cache* [107].

Neste trabalho, o microcontrolador Atmel®AVR®AT32UC3L016 será utilizado como base. Tal microcontrolador, possui arquitetura RISC (*Reduced Instruction Set Computer*), frequências de até 50Mhz, aritmética em ponto-fixo e suporte à instruções 32-*bit*. Além disso, a metodologia possui suporte ao microcontrolador MSP430G2231, de 16-*bits* e 16Mhz, e também a outros processadores com base nas especificações do usuário.

Verificação de Estabilidade

A estabilidade é um requisito fundamental durante o projeto de controladores digitais. Desta forma, um sistema discreto é estável se todos os seus polos estão na região interior do círculo unitário do plano z (*i.e.*, os polos devem ter módulo menor que um) [108].

Em estudos anteriores com o verificador ESBMC [39, 40], a verificação de estabilidade era feita utilizando decomposição de Schur, com a biblioteca *Eigen* [109]. No entanto, esse método utiliza várias operações matriciais, o qual torna o processo custoso computacionalmente. Neste trabalho, foi adotado o teste de estabilidade de Jury [41], por ter um algoritmo menos custoso que a decomposição de Schur e por não necessitar de uma biblioteca externa, sendo capaz de ser executado simbolicamente pelo verificador de modelos. Sua complexidade é $O(n^2)$ (quadrática), ao contrário de Schur que é $O(n^3)$ (cúbica) [109].

O algoritmo de Jury pode ser aplicado a um polinômio na forma:

$$F(z) = a_n z^n + a_{n-1} z^{n-1} + ... a_1 z + a_0 = 0, a_n > 0, \tag{3.9}$$

onde a_n até a_0 representa os coeficientes do denominador do controlador digital. Em especial, estes coeficientes são distribuídos na tabela de Jury usando o formato mostrado na Tabela 3.1.

linha	z^n	z^{n-1}	...	z^{n-k}	...	z^1	z^0
1	a_n	a_{n-1}	...	a_{n-k}	...	a_1	a_0
2	a_0	a_1	...	a_k	...	a_{n-1}	a_n
3	b_0	b_1	...	b_{n-k}	...	b_{n-1}	0
4	b_{n-1}	b_{n-2}	...	b_k	...	b_0	0
5	c_0	c_1	...	...	c_{n-2}	0	0
6	c_{n-2}	c_{n-3}	...	...	c_0	0	0
...	...	...	...	...	...	...	...
$2n-1$	r_0	0	0	0	0	0	0

Tabela 3.1: Generalização dos coeficientes do denominador de um controlador digital distribuídos em uma tabela de Jury.

Considerando a tabela de Jury como uma matriz m com dimensões $[2n-1][n]$ onde n é o número de coeficientes. Algumas considerações são feitas para o algoritmo:

a) A primeira linha da matriz m é composta pelos coeficientes do denominador do controlador digital.

b) As linhas pares têm os coeficientes da linha anterior dispostos em ordem inversa (*i.e.*, descartando os zeros do final).

c) O b_0 está na linha 3 da coluna 1 e seu valor é:
$b_0 = m[3-2][1] - (m[3-2][p]/m[3-2][1]) * m[3-1][1]$.

Generalizando b_j e c_j é formado por $m[i][j] = m[i-2][j] - (m[i-2][p]/m[i-2][1]) * m[i-1][j]$, onde p é a ultima coluna com valor não nulo para a linha $i-2$.

d) A linha $2n-1$ tem apenas um elemento não nulo na primeira coluna.

com a tabela de Jury preenchida, é necessário verificar a estabilidade usando as seguintes definições:

Definição 2 *Se o elemento $m[1][1]$ é positivo, então $F(z)$ será estável, se e somente se, todos os primeiros elementos nas linhas ímpares (i.e., $m[3][1]$, $m[5][1]$, ..., $m[2n-1][1]$) são positivos também.*

Definição 3 *Se o elemento $m[1][1]$ é negativo, então $F(z)$ será estável, se e somente se, o primeiro elemento das linhas ímpares (i.e., $m[3][1]$, $m[5][1]$, ..., $m[2n-1][1]$) são negativos também.*

Como exemplo, será considerado a verificação de estabilidade para o seguinte controlador digital (retirado dos nossos casos de teste, ver Apêndice B):

$$H(z) = \frac{-0.4603z^2 + 1.006z - 0.5421}{z^2 + 0.5949z - 0.3867}. \tag{3.10}$$

Este controlador digital tem sua distribuição de Jury mostrada na Tabela 3.2.

linha	z^2	z^1	z^0
1	1.0	0.5949	−0.3867
2	−0.3867	0.5949	1.0
3	0.8505	0.8249	0
4	0.8249	0.8505	0
5	0.0503	0	0

Tabela 3.2: Tabela de Jury para o controlador digital representado pela Equação (3.10) usando os coeficientes do denominador.

Analisando a Tabela 3.2 e considerando a Definição 2, pode-se facilmente concluir que o controlador digital representado pela Equação (3.10) é estável, uma vez que $m[1][1]$, $m[3][1]$, e $m[5][1]$ são números positivos.

A condição para verificação de estabilidade de acordo com o critério de Jury é representado pelo literal $l_{stability}$, que possui a seguinte restrição:

$$\begin{aligned} l_{stability} \Longleftrightarrow ((sgn(m[i][1]) \neq sgn(m[i+2][1])) \vee \\ (sgn(m[i][1]) = sgn(m[i+2][1]))), \forall i, \end{aligned} \tag{3.11}$$

onde $sgn(\bullet)$ é uma função que indica o sinal do operando.

Verificação de Fase Mínima

Um sistema de fase mínima é definido por um sistema estável com todos os seus zeros estáveis. Conceitualmente, um sistema de fase mínima tem todos os seus zeros dentro do círculo unitário [41]. Tal propriedade é desejável nos controladores digitais devido aos sistemas em malha fechada.

A verificação de fase mínima é similar a utilizada para a verificação de estabilidade. No entanto, ao invés de utilizar os coeficientes do denominador do controlador digital, para checar a fase mínima, são utilizados os coeficientes do numerador. Como exemplo, será checado se o controlador digital representado pela Equação (3.10) tem fase mínima. A tabela de Jury usando os coeficientes do numerador para esse controlador digital é mostrada na Tabela 3.3:

linha	z^2	z^1	z^0
1	-0.4603	1.006	-0.5421
2	-0.5421	1.006	-0.4603
3	0.1781	-0.1788	0
4	-0.1788	0.1781	0
5	-0.0013	0	0

Tabela 3.3: Tabela de Jury para o controlador digital representado pela Equação (3.10) usando os coeficientes do numerador.

Analisando a Tabela de Jury 3.3 e considerando a Definição 2, pode-se concluir que o controlador digital representado pela Equação (3.10) não tem fase mínima, uma vez que $m[1][1]$

e $m[5][1]$ são negativos, no entanto, $m[3][1]$ é positivo. Note que a verificação de fase mínima é similar a verificação de estabilidade.

Verificação de Estabilidade e Fase Mínima Para os Controladores da Forma Delta

O critério de Jury é um método poderoso para verificar a convergência de polinômios em tempo discreto no domínio z. No entanto, esse critério não fornece garantia para o domínio *delta*. Por esse motivo, são estabelecidos critérios diferentes para a verificação de estabilidade dos controladores implementados na forma delta. Estes critérios são representados pelas Definição 4 e Lema 2:

Definição 4 *Um sistema linear dinâmico, representado pela Equação (2.4), de ordem N, é assintoticamente estável, se e somente se, seus polos ($\lambda_1, \lambda_2, ..., \lambda_N$) são uma sequência convergente que satisfaz a propriedade (a prova é fornecida por Feue e Goodwin [110]):*

$$lim_{k\to\infty} \sum_{i=1}^{N} c_i(1+\Delta\lambda_i)^k = 0, \tag{3.12}$$

onde c_i é um coeficiente complexo não nulo.

Analisando a Equação (3.12), pode-se concluir que a soma é convergente, se e somente se, a parte exponencial apresenta convergência. Assim, a estabilidade é verificada como:

Lema 2 *Um sistema digital na forma delta modelado pela Equação (2.4) é assintoticamente estável, se e somente se, todos os seus autovalores $\lambda_i, i = 1, 2, ...N$ (simples ou múltiplos) satisfazem:*

$$|1+\Delta\lambda_i| < 1. \tag{3.13}$$

onde Δ corresponde ao coeficiente de otimização utilizado na transformação. Sendo assim, a mesma ideia pode ser aplicada na verificação de fase mínima, desde que seja utilizado o polinômio do numerador.

3.1.2 Verificação de Controladores Digitais em Malha Fechada

A Figura 3.8 mostra uma visão complementar da metodologia apresentada na Seção 3.1, para verificação de controladores e plantas em malha fechada utilizando o DSVerifier. É incluído um novo passo (Passo 4) responsável pela construção do modelo não determinístico da

planta. Neste passo, o usuário informa os coeficientes do numerador e do denominador da planta, e também o percentual de variação para cada um destes coeficientes. Atualmente, a única propriedade suportada pela ferramenta é a estabilidade em malha fechada considerando incertezas paramétricas. O DSVerifier checa a propriedade desejada e no passo 7 retorna o resultado da verificação, que pode ser: sucesso (caso não haja nenhuma violação na propriedade), ou caso contrário, a ferramenta retorna falha e mostra um contraexemplo com a variação dos coeficientes da planta, demonstrando a margem de estabilidade. Um exemplo de utilização da metodologia é fornecida no Apêndice A.

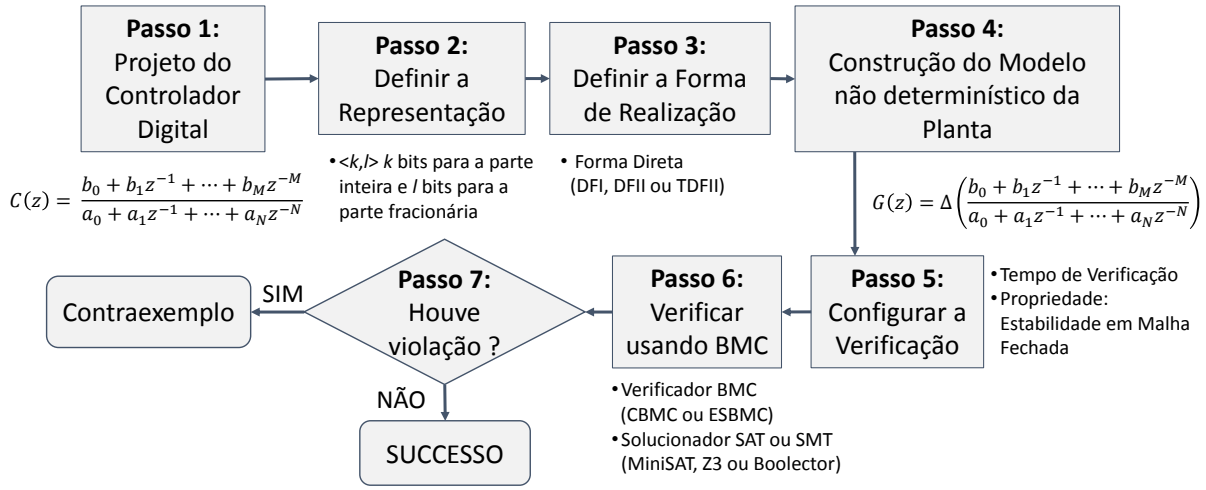

Figura 3.8: Metodologia proposta para verificação de controladores digitais e plantas em malha fechada.

Verificação de Estabilidade Considerando Incertezas Paramétricas

Para a verificação de estabilidade em malha fechada, o DSVerifier recebe como entrada a função de transferência do controlador digital, informações a respeito da sua implementação que compõem o formato FWL (*i.e.*, número de *bits* da parte inteira, parte fracionária, e respectiva forma de realização) e também a função de transferência do modelo da planta. Suponha que o controlador digital $C(z)$ e o modelo da planta $G(z)$ sejam respectivamente as equações descritas abaixo:

$$C(z) = \frac{\beta_0 + \beta_1 z^{-1} + ... + \beta_{M_C} z^{-M_C}}{\alpha_0 + \alpha_1 z^{-1} + ... + \alpha_{N_C} z^{-N_C}}, \quad G(z) = \frac{b_0 + b_1 z^{-1} + ... + b_{M_G} z^{-M_G}}{a_0 + a_1 z^{-1} + ... + aN_G z^{-N_G}}. \tag{3.14}$$

Seus coeficientes podem ser arranjados nos vetores c_0 e p_0:

$$c_0 = [\beta_0 \quad \beta_1 z^{-1} \quad ... \quad \beta_{M_C} z^{-M_C} \quad \alpha_0 \quad \alpha_1 \quad \alpha_{N_C}], \tag{3.15}$$

$$p_0 = [b_0 \quad b_1 z^{-1} \quad ... \quad b_{M_C} z^{-M_G} \quad a_0 \quad a_1 \quad a_{N_G}]. \tag{3.16}$$

Para cada implementação de $C(z)$ existe uma função $FWL[\cdot] : \mathbb{R}^{N+M+2} \rightarrow Q[\mathbb{R}^{N+M+2}]$ que aplica os efeitos FWL no sistema digital, onde $Q[\mathbb{R}]$ representa o conjunto quantizado dos valores reais, definidos pelos parâmetros da implementação.

A incerteza da planta pode ser expressa como:

$$p = p_0 + \Delta p = [b_0 + \Delta b_0 \quad b_1 + \Delta b_1 \quad ... \quad b_{M_G} + \Delta b_{M_G} \quad a_0 + \Delta a_0 \quad a_1 + \Delta a_1 \quad ... \quad a_{N_G} + \Delta a_{N_G}], \tag{3.17}$$

onde Δp representa as incertezas da planta e $\Delta p\%$ corresponde ao percentual de variação nos coeficientes de p_0. Considerando $0 \leq i \leq M_G$ e $0 \leq j \leq N_G$ temos:

$$\|\Delta b_i\| \leq \frac{\Delta b_i\% \cdot b_i}{100}, \tag{3.18}$$

$$\|\Delta a_j\| \leq \frac{\Delta a_j\% \cdot a_i}{100}, \tag{3.19}$$

$$\begin{aligned} \Delta p\% \quad = \quad & \big[\Delta b_0\% \quad \Delta b_1\% \quad ... \quad \Delta b_{M_G}\% \\ & \Delta a_0\% \quad \Delta a_1\% \quad ... \quad \Delta a_{N_G}\%\big]. \end{aligned} \tag{3.20}$$

A partir das entradas c_0, p_0 e $\Delta p\%$ fornecidas pelo usuário, o polinômio característico para verificação de estabilidade em malha fechada (apresentado na Seção 2.4) é gerado considerando valores não determinísticos (*i.e.*, cada parâmetro da planta pode assumir todos possíveis valores no intervalo do seu percentual de incerteza).

O polinômio característico tem a sua estabilidade analisada pelo critério de Jury apresentado na Seção 3.1.1. Caso ocorra uma violação na propriedade de estabilidade, o DSVerifier retorna um contraexemplo que contém a combinação de coeficientes da planta que levarão para a falha.

Como exemplo, considere o controlador descrito pela Equação (3.21) e sua respectiva planta (3.22) (retirados de [1, 103]), ambos estão discretizados com período de amostragem de 0.05 segundos.

$$C_4(z) = \frac{14.2344952240432z - 13.5523885821810}{z - 1.42685562045612} \tag{3.21}$$

$$G_1(z) = \frac{0.0500422033454653z - 0.052606826445634}{z^2 - 2.05640034257636z + 1.05127109637602} \tag{3.22}$$

Considerando que o controlador seja implementado utilizando 4 *bits* de precisão, inicialmente, essa configuração é estável. No entanto, caso seja adicionada uma incerteza de $\pm 0.25\%$ em todos os parâmetros da planta, o DSVerifier encontra uma violação para as seguintes variações paramétricas:

$$\Delta p\% = [\quad 0.1229 \quad 0.2167 \quad 0.0441 \quad -0.2499 \quad 0.1026 \quad]. \tag{3.23}$$

3.2 Resumo

Neste capítulo, foi apresentada a metodologia para verificação de controladores digitais implementados em processadores de ponto-fixo suportada pelo verificador de sistemas digitais (DSVerifier). Tal metodologia inclui propriedades que podem ser conectadas e verificadas em diferentes verificadores de modelos limitados (*e.g.,* CBMC ou ESBMC), dentre elas, a verificação de estouros aritméticos, que utiliza valores não determinísticos (*i.e.*, todos os possíveis valores dentro de um intervalo dinâmico informado pelo usuário), afim de encontrar saídas que estejam fora do intervalo da palavra finita. Foi abordada também, a verificação de oscilações de ciclo limite considerando entradas constantes não determinísticas, afim de encontrar violações causadas por estouros aritméticos tratados via *wrap-around*, ou ainda decorrentes da precisão finita definida pelo usuário. A metodologia também fornece suporte à verificação de restrições temporais, que permite o balanceamento entre a representação em ponto-fixo e os requisitos de *hardware*. Além disso, é possível verificar a estabilidade, fase mínima, e ainda a estabilidade em malha fechada considerando variações paramétricas definidas pelo usuário. Como resultado, o conteúdo apresentado nesse capítulo fornece todo o embasamento necessário para compreensão dos resultados obtidos na avaliação experimental, que será descrito na próxima seção.

Capítulo 4

Avaliação Experimental

Nesse capítulo, será descrita a avaliação experimental do estudo proposto. Primeiramente, a Seção 4.1 descreve os objetivos dos experimentos. Em seguida, na Seção 4.2 é descrita a sua configuração. A Seção 4.3, descreve a verificação de controladores considerando as propriedades de estouros aritméticos, ciclo limite, restrições temporais, estabilidade e fase mínima. Além disso, a Seção 4.4 descreve a verificação de estabilidade em malha fechada. Concluindo, a Seção 4.5 resume os resultados experimentais e suas abordagens.

4.1 Objetivos dos Experimentos

- Demonstrar que a metodologia para verificação de controladores digitais implementados em ponto-fixo suportada pelo DSVerifier, é capaz de encontrar falhas decorrentes das etapas de projeto.
- Avaliar o desempenho das realizações nas formas diretas e delta, mediante as propriedades descritas na Seção 3, levando em consideração diferentes valores de precisão.
- Aplicar a metodologia para a verificação de controladores digitais em malha fechada, baseado em controladores e plantas da literatura.
- Comparar o desempenho do DSVerifier, utilizando os verificadores de modelos CBMC (baseado em satisfação booleana) e ESBMC (baseado em teorias do módulo da satisfatibilidade) com diferentes solucionadores.

4.2 Configuração dos Experimentos

Esse estudo utiliza dois verificadores de modelos como *back-end* para o DSVerifier: o ESBMC v2.0[1] com diferentes solucionadores SMT (*e.g.*, Z3 4.0 e Boolector v2.0.1[2]) e o CBMC v5.1[3] com solucionador SAT MiniSAT v2.2.0.

Para os experimentos de verificação de controladores digitais, são consideradas as cinco propriedades descritas no Capítulo 3, e o desempenho das realizações delta (com diferentes valores de Δ) é comparado ao das formas diretas. Como resultado, todos os casos de teste foram verificados em 9 realizações diferentes que são: Forma Direta I (DFI), Forma Direta II (DFII), Forma Direta II Transposta (TDFII), Forma Direta Delta I (DDFI), Forma Direta Delta II (DDFII) e Forma Direta Delta II Transposta (TDDFII), de modo que cada forma delta foi implementada utilizando os valores $1/2$ e $1/4$.

Foi utilizado o suporte a linha de comando do DSVerifier descrito no Apêndice A, considerando uma plataforma de *hardware* com 32-*bits* e *clock* de 50Mhz, um limite máximo $k = 10$, e o tempo máximo de verificação de 7200 segundos. Em especial, a verificação de estabilidade em malha fechada terá o tempo máximo de 14400 segundos, em virtude do menor número de casos de teste (verificações que ultrapassem esse tempo são consideradas como *time-out*).

Todos os experimentos foram executados em um computador com a seguinte configuração: Processador Intel Core i7 – 2600 3.40 GHz, 24 GB de RAM, com sistema operacional Ubuntu 14.04 LTS 64-bits.

4.3 Verificação de Controladores Digitais

4.3.1 Projeto dos Controladores Digitais

Como base para os experimentos, foram desenvolvidos 20 controladores digitais para quatro plantas diferentes (ver Apêndice B). Desse total, 4 controladores para uma planta comercial de um *ball and beam* que tem o seguinte modelo matemático:

[1]Disponível em: http://esbmc.org
[2]Disponível em: http://fmv.jku.at/boolector/
[3]Disponível em: http://www.cprover.org/cbmc/

$$G_1(z) = \frac{1.0067 \times 10^{-8}(z+9.256)(z+0.9324)}{(z-1)^3(z-0.7041)}, \tag{4.1}$$

onde o período de amostragem é 0.01 segundos. Outros 6 controladores foram projetados para uma planta de um motor A/C extraído do Ogata [104], descrito por:

$$G_2(s) = \frac{1}{s(s+1)}, \tag{4.2}$$

essa planta é discretizada com diferentes taxas de amostragem. Além disso, foram desenvolvidos 7 controladores para outra planta extraída do Ogata [104], seguindo:

$$G_3(s) = \frac{1}{s(s+0.4)}, \tag{4.3}$$

também discretizada em diferentes tempos de amostragem. Por último, 3 controladores foram projetados para a estabilização de um quadrirotor. Tais controladores possuem o período de amostragem de 0.02 segundos e seus modelos dos ângulos de rolagem, arfagem e guinada, seguem

$$G_4(s) = \frac{-0.06875s^2}{s^2 - 1.696s + 0.7089} \quad e \quad G_5(s) = \frac{-0.04785s^2}{s^2 - 1.789s + 0.8014}. \tag{4.4}$$

O formato FWL utilizado nestes experimentos é composto pela parte inteira definida pela metodologia apresentada por Carletta *et al.* [12] e 3 diferentes valores para a precisão, com 4, 8 e 12 *bits*.

4.3.2 Resultados Experimentais

Foram executadas aproximadamente 2500 verificações com os controladores descritos na seção anterior. Tais verificações estão divididas em cinco categorias: estouros aritméticos, ciclo limite (considerando entradas nulas e constantes não determinísticas), estabilidade, fase mínima e restrições temporais. Além disso, foram consideradas como realizações as formas diretas DFI, DFII e TDFII, e também seus correspondentes nas formas delta, com diferentes valores de otimização (delta $1/2$, onde $\Delta = 0.5$ e delta $1/4$, onde $\Delta = 0.25$).

Verificação de Estouros Aritméticos

A Figura 4.1 mostra os resultados da verificação de estouros aritméticos para as realizações nas formas diretas e delta.

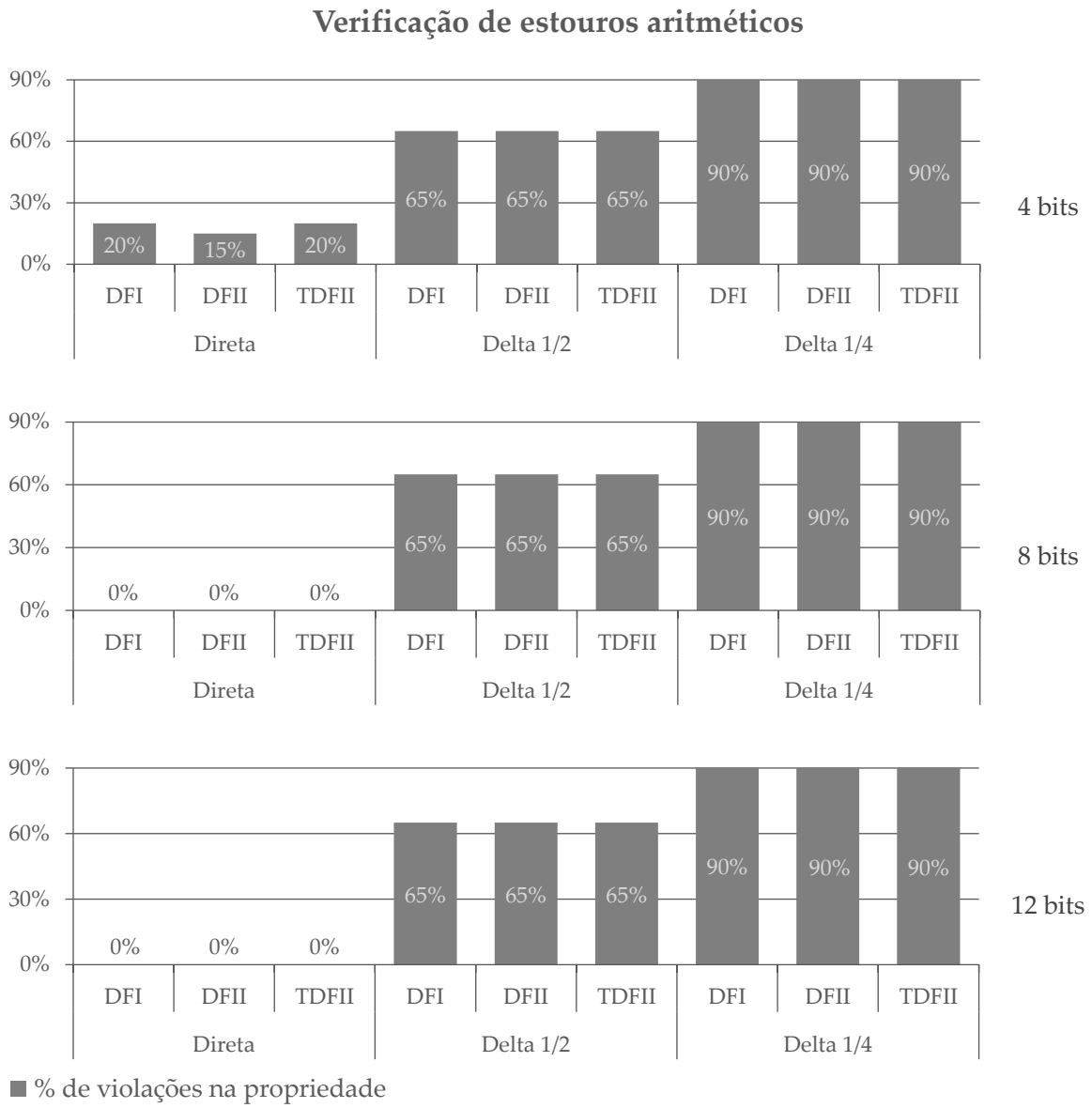

Figura 4.1: Resultados da verificação de estouros aritméticos para as realizações nas formas diretas e delta, com diferentes valores de precisão.

Nesta figura são apresentados três gráficos, um para cada valor de precisão, com 4, 8 e 12 *bits*, respectivamente. Em cada gráfico, o eixo horizontal representa as realizações (*i.e.*, Direta, Delta 1/2 e Delta 1/4), subdividas em três diferentes formas: DFI, DFII e TDFII. O eixo vertical apresenta o percentual de violações (%) encontrado pela verificação para cada forma, realização e implementação. No geral, para cada valor de precisão foram executados 180 testes

(*i.e.*, 60 para formas diretas, 60 para Delta $1/2$ e 60 para a Delta $1/4$).

Com base nos resultados, pode-se notar que o DSVerifier foi capaz de detectar violações nas formas delta, e também na formas diretas com 4 *bits* de precisão, mesmo utilizando um critério conservador para definição do formato FWL. Para tal propriedade, considerando 4 *bits* de precisão, o DSVerifier encontrou violações em até 20% dos casos de teste na forma direta. Utilizando $\Delta = 1/2$ o número de violações subiu para 65%. E se considerarmos $\Delta = 1/4$, esse percentual aumenta para 90% (*i.e.*, 70% mais violações que as formas diretas). Além disso, considerando 8 e 12 *bits*, não houve violações de estouro aritmético nas formas diretas e o desempenho das formas delta se manteve. Com relação às realizações diretas em 4 *bits* de precisão, observou-se que as falhas na propriedade ocorreram devido aos efeitos FWL (decorrentes da quantização) sobre os coeficientes dos controladores digitais, afetando os seus polos, e levando o sistema para a instabilidade. No geral, as formas diretas apresentaram melhores desempenhos devido ao formato FWL definido por Carletta *et al.* [12], que considera o domínio z e não o domínio delta.

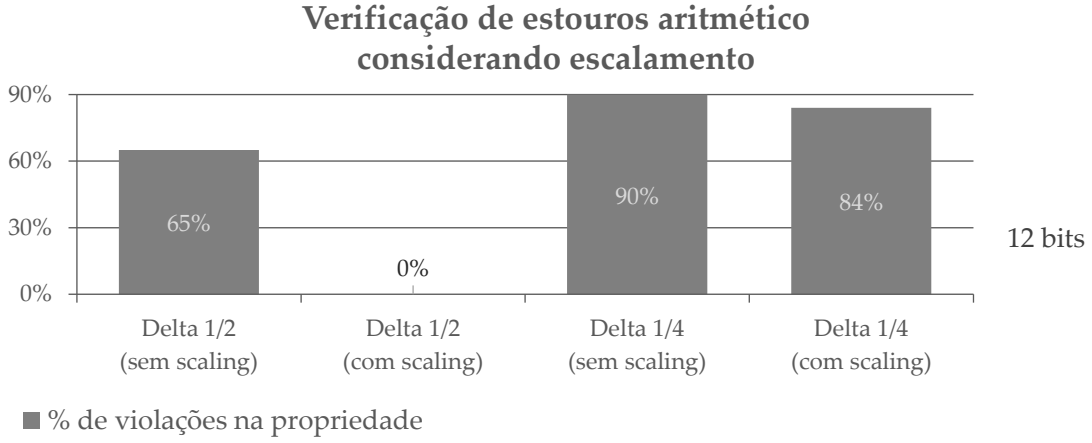

Figura 4.2: Resultados da verificação de estouros aritméticos na forma delta, considerando *scaling*.

Por meio da utilização de escalamento dos valores de entrada (*scaling*) é possível contornar e eliminar os estouros aritméticos. Afim de melhorar os resultados após a otimização com o operador *delta*, foram considerados o uso de diferentes fatores de escala, variando entre 10 e 10000, o resultado produzido pode ser visto na Figura 4.2. Para $\Delta = 1/2$, o número de violações reduziu de 65% para 0%. No entanto, utilizando o mesmo valor de *scaling* para $\Delta = 1/4$ ainda foi possível observar violações na propriedade. Com isso pode-se perceber que quanto menor o valor de delta, maior deve ser o escalamento, ou o número de *bits* definidos no

formato FWL. Além disso, é importante ressaltar que o fator de escala afeta diretamente a razão sinal-ruido, modificando as saídas produzidas pelo controlador digital. E também que escalamentos excessivos (*i.e.,* sem considerar o formato FWL), podem ocasionar arredondamentos indesejados para zero nos valores de entrada.

Verificação de Ciclo Limite

Para a propriedade de ciclo limite, a verificação foi dividida em duas etapas, inicialmente considerando entradas constantes nulas (*i.e.,* zeros), seguida da verificação considerando constantes não deterministicas, afim de comparar os seus resultados.

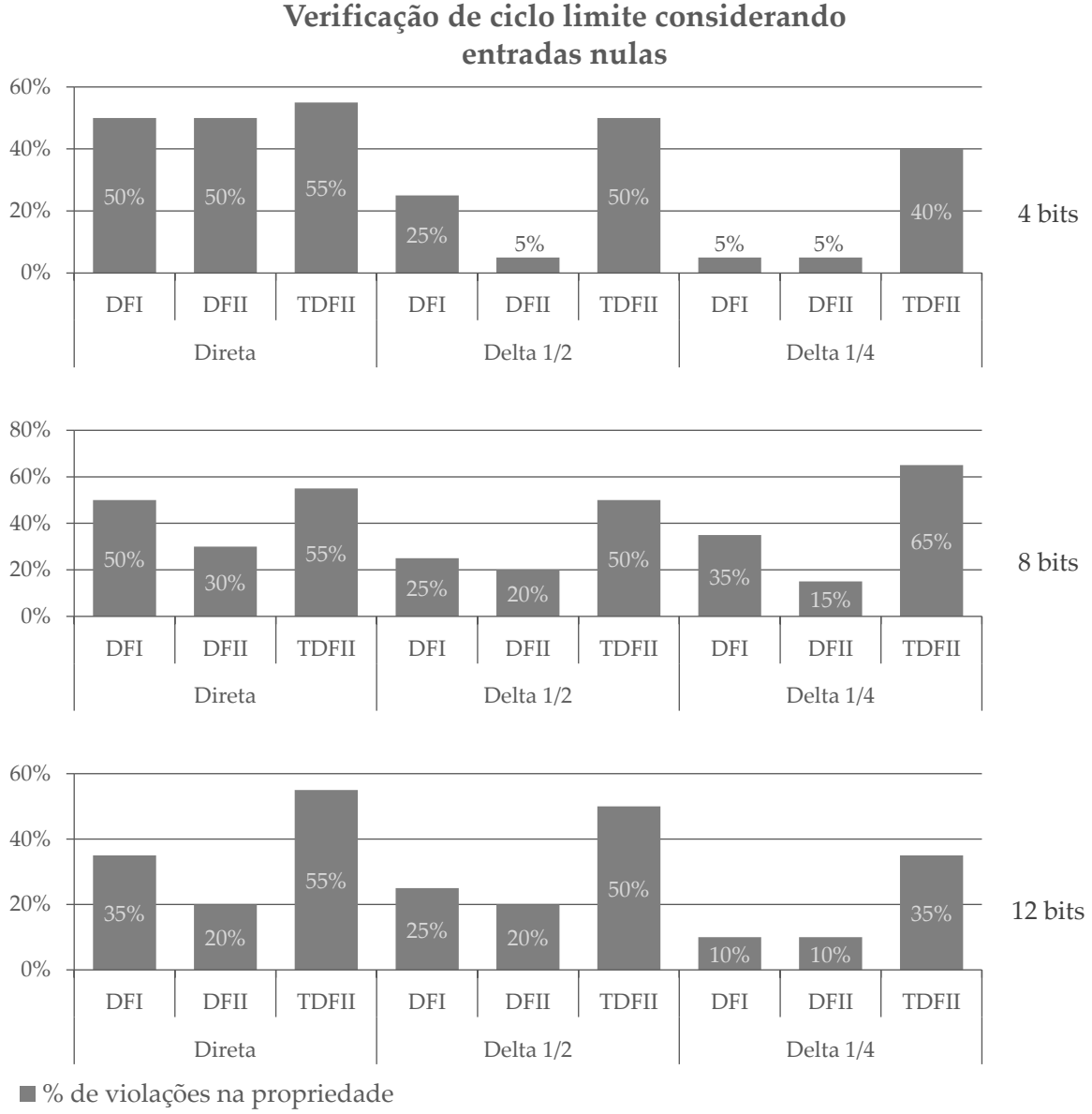

Figura 4.3: Resultados da verificação de ciclo limite considerando apenas entradas nulas.

Os resultados das verificações de ciclo limite considerando entradas nulas podem ser vistos na Figura 4.3. Como pode-se perceber, para 4 *bits* de precisão, o DSVerifier encontrou violações em até 55% dos casos de teste das realizações diretas. Dentre tais verificações, os piores resultados foram apresentados pela Forma Direta II Transposta (TDFII). Aplicando $\Delta = 1/2$, foi possível reduzir o número de violações da Forma Direta II (DFII) para 5% (uma redução de 45% no número de falhas). Considerando $\Delta = 1/4$, o número de violações apresentou resultados semelhantes. Além disso, após efetuar a verificação de ciclo limite com entradas nulas para 8, e 12 *bits* de precisão, pode-se perceber que as formas diretas continuaram apresentando violações em até 55% dos casos de teste. E após otimizar os coeficientes com $\Delta = 1/2$ e $\Delta = 1/4$, também foi possível observar melhores resultados quando comparados às formas diretas.

A utilização de entradas constantes não determinísticas para a verificação de ciclo limite também é proposta neste trabalho. Os resultados gerados pelo DSVerifier para essa configuração são mostrados na Figura 4.4. Inicialmente, para 4 *bits*, o DSVerifier foi capaz de encontrar violações em até 65% dos casos de teste das realizações na Forma Direta II. Após aplicar a otimização com $\Delta = 1/2$, foi possível reduzir o número de falhas em até 20% para tal forma. Além disso, considerando $\Delta = 1/4$, foi possível reduzir aproximadamente 25% das violações. Em geral, para a precisão de 4 *bits*, as formas delta apresentaram melhores resultados se comparados às formas diretas. Para a precisão de 8 e 12 *bits*, houve casos onde o desempenho das formas delta foram superiores às formas diretas, no entanto também é possível notar casos onde as formas diretas apresentaram melhor desempenho se comparado às formas delta. Como por exemplo, a Forma Direta I (DFI) otimizada com $\Delta = 1/4$ em 8 *bits*, que teve um desempenho pior que as suas concorrentes. E também a Forma Direta II (DFII) implementada com 12 *bits*, que apresentou menos violações que as suas correspondentes otimizadas com o operador *delta*. O maior índice de violações após otimização com $\Delta = 1/2$ e $\Delta = 1/4$ está relacionado à ocorrência de estouros aritméticos (como apresentado nos resultados propriedade anterior), visto que acontecem em maior proporção (*i.e.,* devido às entradas constantes não determinísticas), sendo tratados via *wrap-around*, que é um possível causador das oscilações de ciclo limite.

Em geral, nota-se que a Forma Direta II (DFII) apresentou menor índice de violações, seja com os coeficientes diretamente embarcados, ou ainda otimizados com o operador *delta*. Também é possível perceber que a Forma Direta II Transposta (TDFII) apresentou os piores resultados no comparativo. Ambos os fatos estão relacionados às suas estruturas discretas

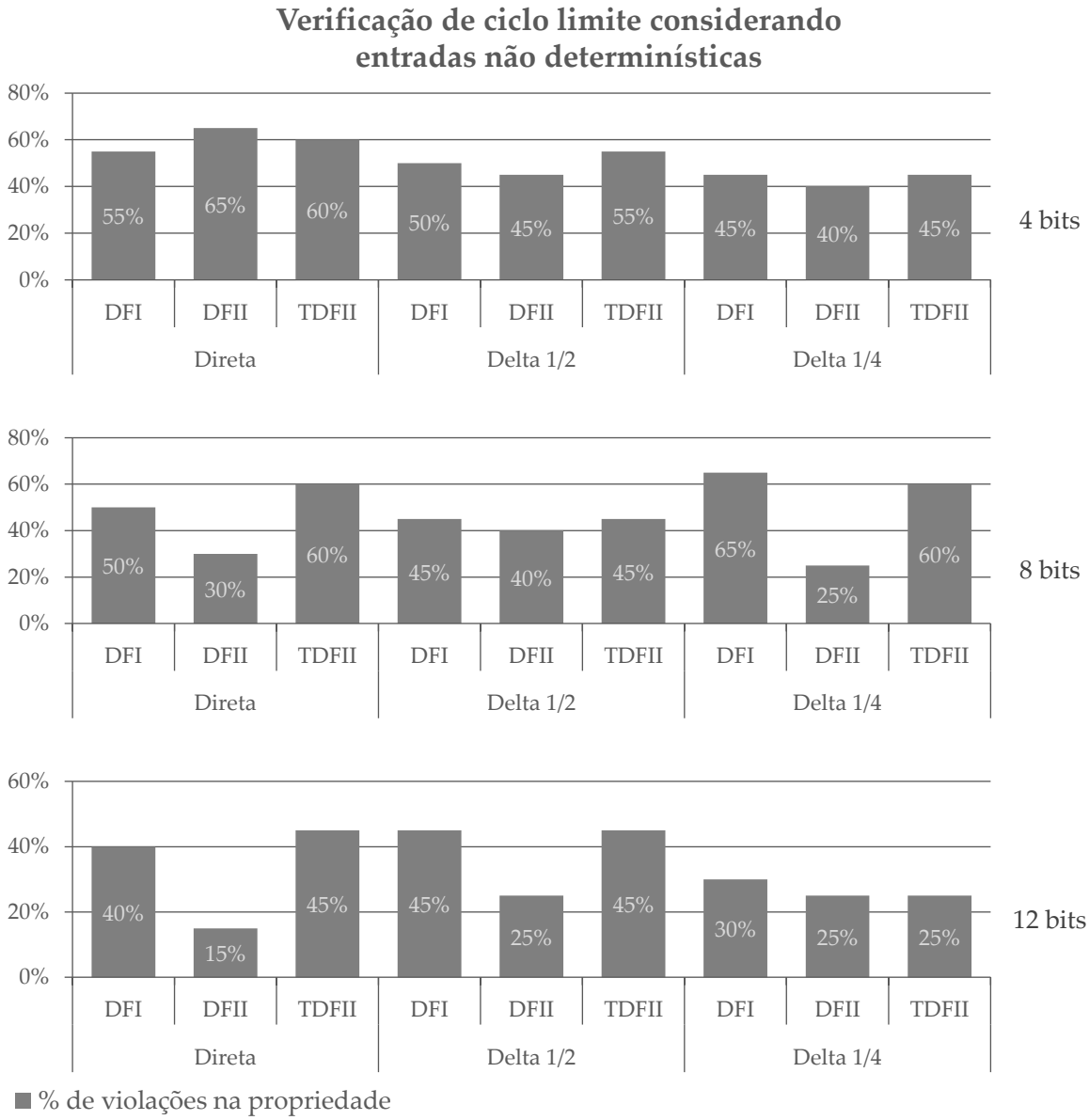

Figura 4.4: Resultados da verificação de ciclo limite considerando constantes não determinísticas.

(*i.e.,* posicionamento dos somadores, multiplicadores e memórias), que alteram a sequência das operações matemáticas e consequentemente a resposta do controlador. Além disso, para tal propriedade, as formas delta apresentaram melhores resultados devido às suas propriedades numéricas, que auxiliam na representação dos coeficientes e ocasionam menores erros de arredondamento em um ambiente de precisão finita, como citado por Middleton e Goodwin [6].

O número total de violações encontradas na verificação de ciclo limite utilizando apenas entradas nulas e o acumulado considerando entradas não determinísticas pode ser visto na Figura 4.5. Considerando apenas entradas nulas, o DSVerifier encontrou violações em 33% dos

casos de teste. No entanto, utilizando entradas não determinísticas, a metodologia foi capaz de encontrar violação em 47% dos casos de teste (*i.e.*, 14% de falhas não foram encontradas com a utilização de entradas nulas).

O percentual de *time-outs* para esta propriedade pode ser visto na Figura 4.6. De modo geral, a verificação de ciclo limite utilizando valores não determinísticos mostrou-se mais custosa computacionalmente devido ao maior espaço de estados a serem explorados, apresentando um número maior de *time-outs* para todos os valores de precisão considerados no experimento.

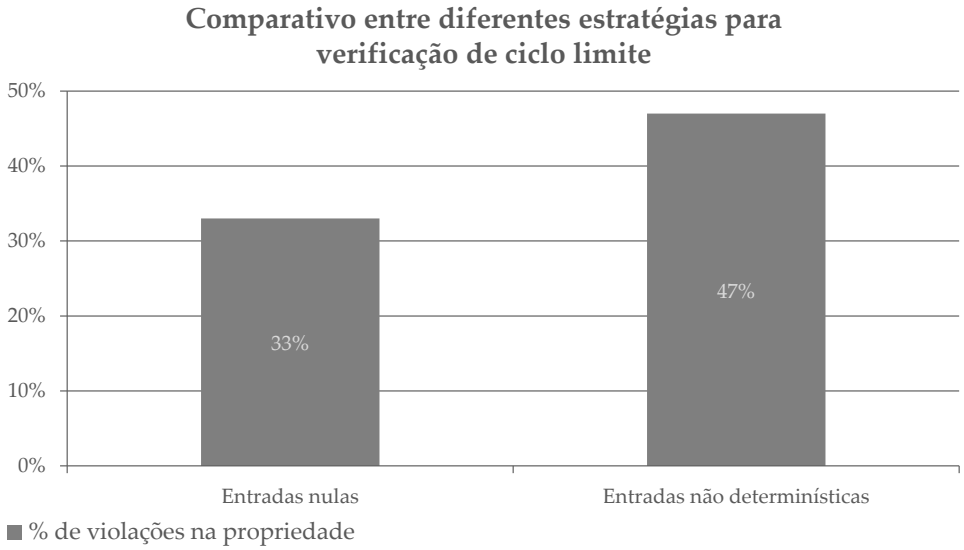

Figura 4.5: Comparativo entre o número de violações de ciclo limite encontradas utilizando diferentes valores de entrada.

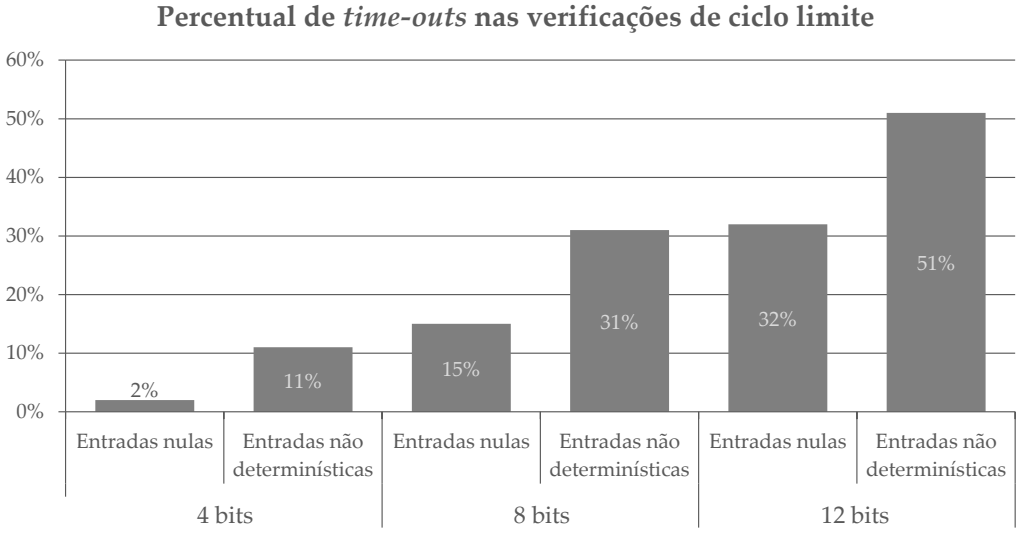

Figura 4.6: Número de *time-outs* nas verificações de ciclo limite.

Verificação de Estabilidade

A Figura 4.7 mostra os resultados da verificação para a propriedade de estabilidade. O eixo horizontal representa as realizações do controlador agrupadas por diferentes valores de precisão, com 4, 8 e 12 *bits*, respectivamente. O eixo vertical mostra o percentual de sucessos e falhas na propriedade. Diferentemente das figuras anteriores, esse gráfico não produz resultados para cada forma, pois eles são baseados apenas nos coeficientes.

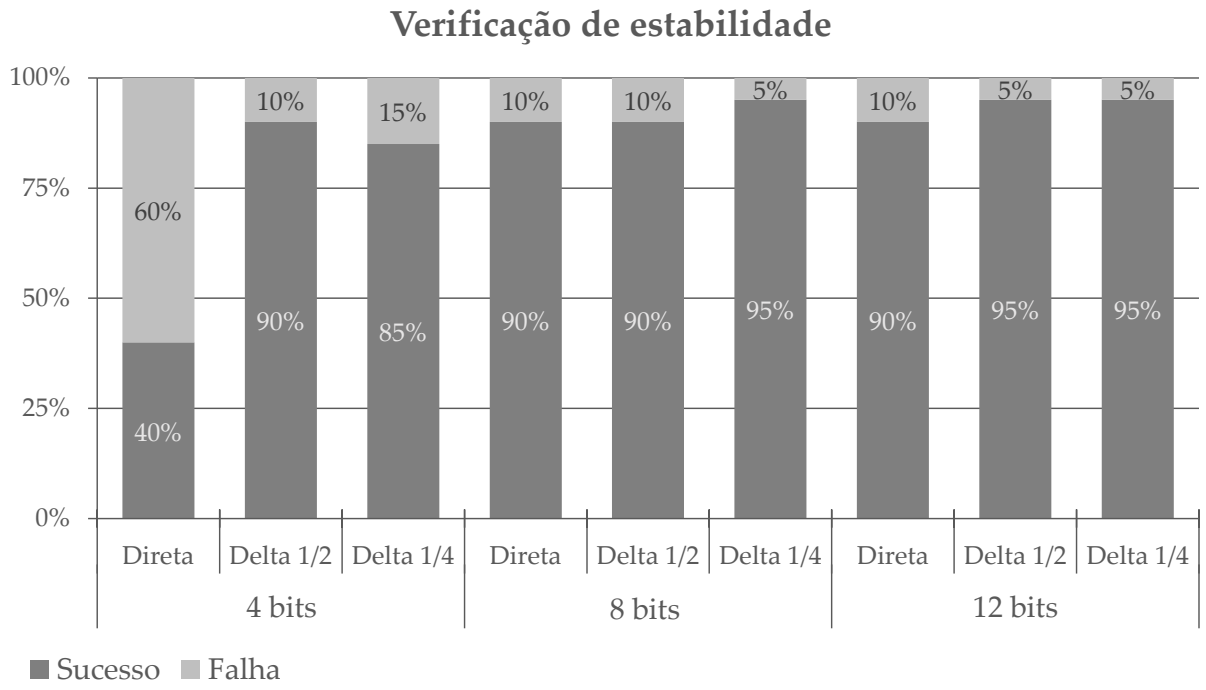

Figura 4.7: Resultados da verificação de estabilidade para os coeficientes da forma direta e delta.

De acordo com os resultados, utilizando 4 *bits* de precisão, os coeficientes na forma direta apresentaram instabilidade em 60% dos casos de teste. Com os coeficientes otimizados utilizando $\Delta = 1/2$, houve falhas em apenas 10% dos controladores (uma redução de 50% no número de violações) e com $\Delta = 1/4$, o percentual de falhas foi de 15%. Em geral, as violações na forma direta implementada com 4 *bits* acontecem devido a baixa precisão, que ocasiona o arredondamento e truncamento dos coeficientes, afetando diretamente o posicionamento dos polos do sistema digital.

Como pode ser visto na Figura 4.8, o controlador 2 (ver Apêndice B) possui inicialmente seus polos em $(0.9815 \pm 0.1584$ e $0.6070)$, e após a quantização com 4 *bits*, os polos se tornam 1.46670 e $0.57914 \pm 0.30118i$, instabilizando o controlador. O mesmo controlador implementado utilizando $\Delta = 1/4$ é estável segundo o critério definido na Seção 3.1.1.

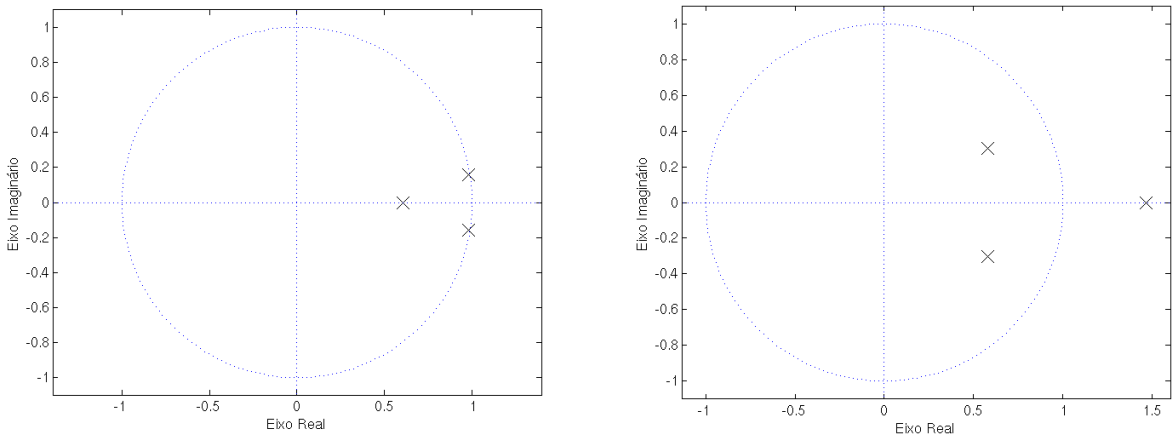

Figura 4.8: Efeito da quantização sobre os polos do controlador 2 (ver Apêndice B).

Além disso, para tal propriedade, com 8 e 12 *bits* de precisão é possível observar violações em 10% dos controladores na forma direta, e em ambos os casos a forma delta apresentou menor índice de violações. Em geral, a forma delta apresenta melhores resultados para a propriedade à medida que o coeficiente de otimização tende para zero, pois sua região de convergência (estabilidade) é amplificada.

Verificação de Fase Mínima

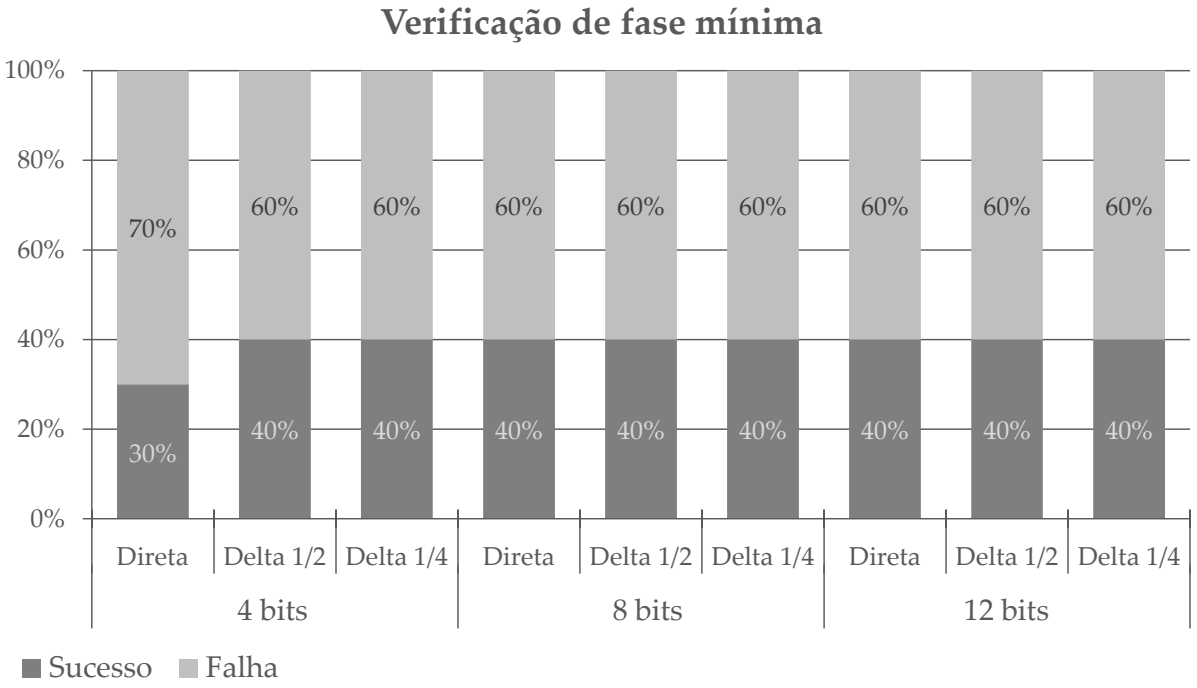

Figura 4.9: Resultados da verificação de fase mínima para as formas diretas e delta.

Os resultados da verificação de fase mínima para os coeficientes na forma direta e delta são apresentados na Figura 4.9. Sua análise é semelhante aos resultados de estabilidade, no entanto, utilizando os coeficientes do numerador do controlador digital. Como pode ser visto, utilizando 4 *bits* de precisão. A forma direta apresentou falha em 70% dos casos de teste, na forma delta o percentual de falha foi de 60%. Além disso, o desempenho dos controladores na forma direta e delta, utilizando 8 e 12 *bits*, foi semelhante. Para esta propridade, temos como exemplo o controlador 10 (ver Apêndice B), que utilizando a forma direta possui um zero em 0.9804. Após a quantização dos seus coeficientes com 4 *bits*, o zero move-se para 1.0 e o controlador tem a propriedade de fase mínima violada.

Verificação de Restrições Temporais

A propriedade de restrições temporais não é exibida em nenhum gráfico, pois todos os testes retornaram sucesso, tanto para as formas diretas quanto para as formas delta. Em especial, esses testes não apresentaram falhas devido à ordem dos controladores ser relativamente baixa e o tempo de amostragem ser razoavelmente alto.

Tempo de Verificação

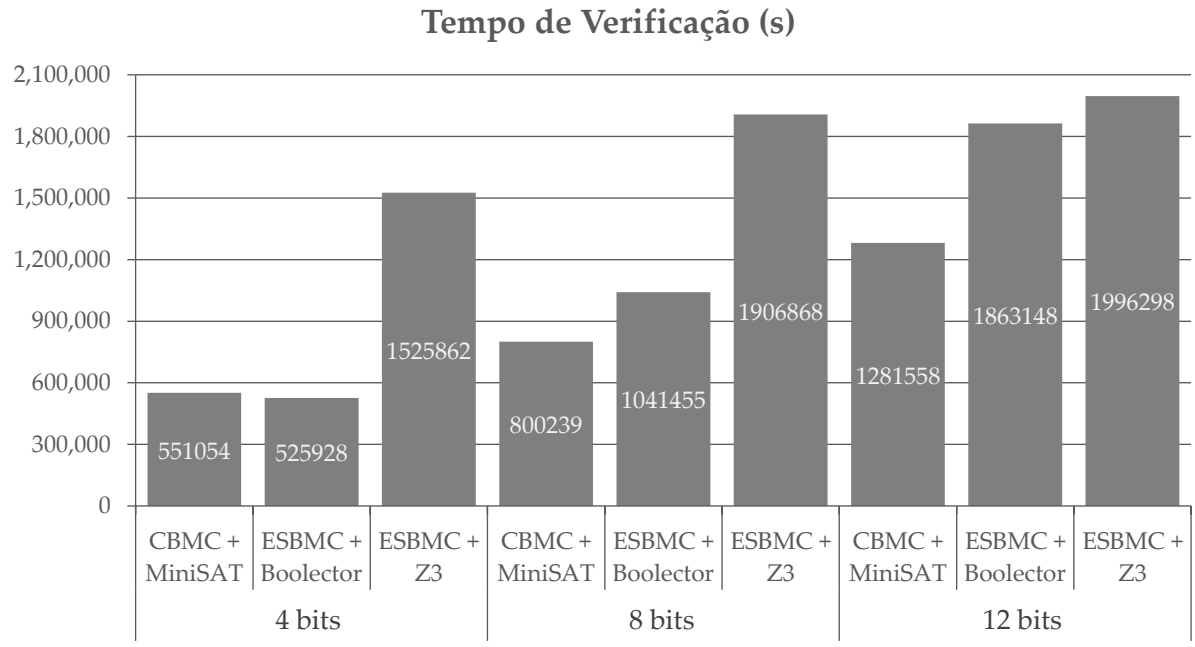

Figura 4.10: Tempo total de verificação considerando diferentes verificadores e solucionadores.

A Figura 4.10 mostra o tempo total de verificação para cada uma das ferramentas utilizadas como *back-end* para o DSVerifier, considerando 4, 8 e 12 *bits* de precisão. Para 4 *bits*, o verificador que apresentou melhor desempenho foi o ESBMC utilizando o solucionador Boolector, com o tempo total de verificação de 525928 segundos. Para 8 e 12 *bits*, o CBMC com solucionador MiniSAT obteve menor tempo de verificação com 800239 e 1281558 segundos, respectivamente. Nota-se que, para as verificações de controladores digitais, o ESBMC utilizando solucionador Z3 v4.0 apresentou o pior desempenho. Em geral, o tempo total gasto por um verificador de modelos limitados leva em consideração a sua construção, técnicas e algoritmos implementados, bem como o seu método de solução. Além disso, solucionadores de satisfação booleana, ou de teorías do módulo de satisfatibilidade, possuem características próprias e contam com diferentes heurísticas que influenciam diretamente no tempo da solução [73].

4.4 Verificação de Controladores Digitais em Malha Fechada

4.4.1 Controladores e Plantas

Como base para os experimentos de malha fechada, foram utilizados dois exemplos apresentados por Keel e Bhattacharyya em [1, 103], que relacionam estabilidade e fragilidade.

O exemplo A, utiliza a função de transferência $G_1(s)$ para a planta, e seu controlador $C_1(s)$:

$$G_1(s) = \frac{s-1}{s^2-s-2}, \quad C_1(s) = \frac{11.44974739s + 11.242640066}{s - 7.03553383}, \tag{4.5}$$

onde ambos são discretizados utilizando o método de Tustin em 9 tempos de amostragem diferentes (*e.g.*, 0.5, 0.1, 0.05, 0.03, 0.01, 0.005, 0.001, 0.00001 e 0.000001 segundos), e implementados usando três diferentes formatos de palavra finita: com 4, 8 e 12 *bits*.

O exemplo B, considera outra planta $G_2(s)$ e seu controlador $C_2(s)$ (retirados de [103]):

$$G_2(s) = \frac{s-1}{s^2+0.5s-0.5}, \quad C_2(s) = \frac{-124.5s^3 - 364.95s^2 - 360.45s - 120}{s^3 + 227.1s^2 + 440.7s + 220}, \tag{4.6}$$

onde ambos são discretizados por meio de Tustin, em 8 diferentes tempos de amostragem (*e.g.*, 0.5, 0.1, 0.05, 0.03, 0.01, 0.005, 0.001 e 0.0004 segundos), também implementados usando três diferentes formatos de palavra finita: com 4, 8 e 12 *bits*.

4.4.2 Resultados Experimentais

Foram executadas aproximadamente 70 verificações com os controladores e plantas descritos na seção anterior. Inicialmente, é considerada a verificação de estabilidade em malha fechada sem incertezas paramétricas, afim de observar os efeitos da palavra finita e tempo de amostragem sobre a estabilidade. A partir dos resultados produzidos, pode-se utilizar a metodologia de verificação proposta na Seção 3.8 nos casos que apresentaram estabilidade.

Exemplo A

Para o exemplo A, o DSVerifier gerou os resultados apresentados na Tabela 4.1, onde "F" descreve os casos de teste que apresentaram instabilidade (*i.e.*, violação na propriedade) e "S" os casos de teste que apresentaram estabilidade.

Tempo de Amostragem (s)	**Formato FWL**		
	4-*bits*	8-*bits*	12-*bits*
0.5	F	F	F
0.1	S	S	S
0.05	S	S	S
0.03	S	S	S
0.01	F	S	S
0.005	F	S	S
0.001	F	F	S
0.00001	F	F	F
0.000001	F	F	F

Tabela 4.1: Verificação de estabilidade em malha fechada para $C_1(s)$ e $G_1(s)$ (4.5), considerando diferentes tempos de amostragem e formatos de palavra finita.

Como pode ser visto na Tabela 4.1, se o controlador e planta forem discretizados com tempo de amostragem de 0.5 segundos, o sistema em malha fechada é instável para todos formatos de palavra finita, neste caso, a taxa de amostragem não é suficiente para o sistema. Além disso, o sistema em malha fechada é estável para 4, 8 e 12 *bits* com os tempos de amostragem 0.1, 0.05 e 0.03 segundos, e à medida que o período de amostragem diminui, o sistema se torna instável. No geral, todas estas considerações são também observadas por Keel e Bhattacharyya em [1, 103].

Considerando que a planta $G_1(s)$ pode variar entre $\pm 0.25\%$ (um percentual informado pelo usuário), a metodologia deve determinar se o sistema em malha fechada ainda será estável.

Após a verificação, temos que: se considerarmos a planta $G_1(s)$ com $\pm 0.25\%$ de incerteza em cada coeficiente e o controlador $C_1(s)$ implementado com 12 *bits* de precisão, ambos discretizados com tempo de amostragem de 0.03 segundos (*i.e.*, um caso de teste que apresentou um bom desempenho anteriormente), haverá uma falha de estabilidade para a seguinte variação paramétrica:

$$\Delta p\% = [\quad -0.18660 \quad -0.06731 \quad -0.21599 \quad 0.06569 \quad 0.06567 \quad]. \tag{4.7}$$

A Figura 4.11 mostra as respostas ao degrau para a $G_1(s)$ e $C_1(s)$ amostrados com 0.03 segundos e implementados com 12 *bits*. Como pode ser visto no quadro esquerdo, sem considerar incertezas nos coeficientes da planta o sistema se mostra estável. Porém, aplicando o contraexemplo indicado pelo DSVerifier (quadro direito), nota-se a instabilidade.

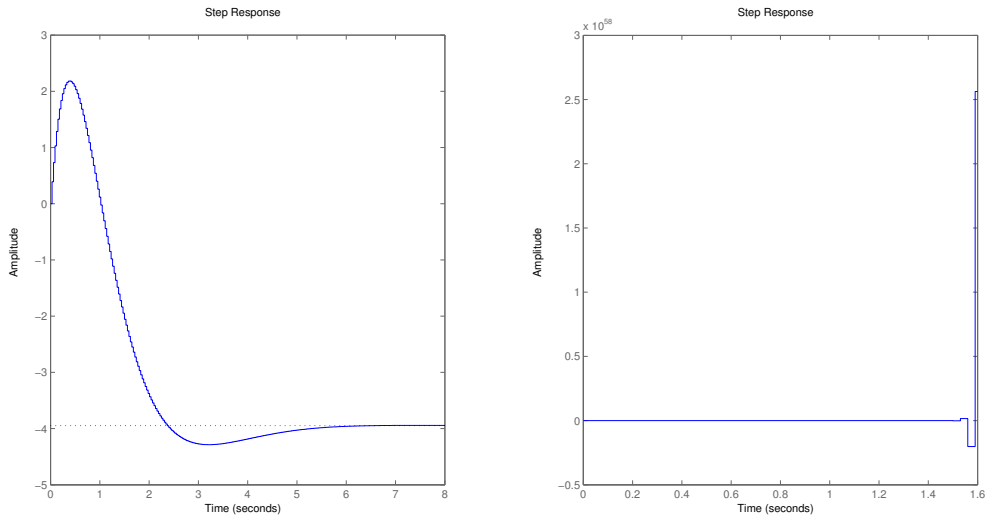

Figura 4.11: Resposta ao degrau para $G_1(s)$ e $C_1(s)$ amostrados com 0.03 segundos. Esquerda: Sistema nominal. Direita: Sistemas com incertezas em $G_1(s)$.

Exemplo B

Utilizando o exemplo B, temos como resultado da verificação de estabilidade em malha fechada (sem considerar incertezas paramétricas) a Tabela 4.2.

A Tabela 4.2 mostra que a variação do número de *bits* e do tempo de amostragem influenciam pouco na estabilidade do sistema em malha fechada. De modo geral, a maioria dos casos de testes apresentam falha de estabilidade. Tais considerações também são feitas em [1, 103].

Tempo de Amostragem (s)	**Formato FWL**		
	4-*bits*	8-*bits*	12-*bits*
0.5	F	S	S
0.1	F	F	S
0.05	F	F	S
0.03	F	F	F
0.01	F	F	F
0.005	F	F	F
0.001	F	F	F
0.0004	F	F	F

Tabela 4.2: Verificação de estabilidade em malha fechada para $C_2(s)$ e $G_2(s)$ (4.6), considerando diferentes tempos de amostragem e formatos de palavra finita.

Neste exemplo, se considerarmos a planta $G_2(s)$ e o controlador $C_2(s)$, ambos amostrados com 0.5 segundos e implementados em 8 *bits*, o sistema em malha fechada apresenta-se estável. No entanto, adicionando uma variação de $\pm 1\%$ nos coeficientes da planta, a ferramenta de verificação retorna que o sistema não será estável caso ocorra a seguinte variação paramétrica:

$$\Delta p\% = [\quad 0.54336 \quad -0.12631 \quad 0.66145 \quad 0.97228 \quad -0.79776 \quad]. \tag{4.8}$$

Como pode ser visto na Figura 4.12, após aplicar a variação paramétrica fornecida pelo contraexemplo do DSVerifier, o sistema se torna instável.

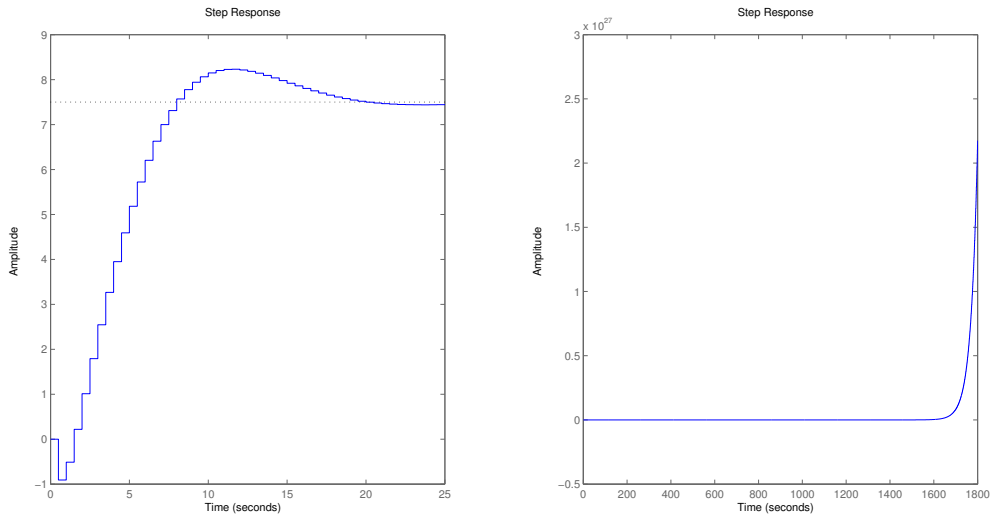

Figura 4.12: Resposta ao degrau para $G_2(s)$ e $C_2(s)$ amostrados com 0.5 segundos. Esquerda: Sistema nominal. Direita: Sistemas com incertezas em $G_2(s)$.

Tempo de Verificação

A Figura 4.13 mostra o tempo total de verificação utilizado pelas ferramentas: CBMC, ESBMC com solucionador Boolector e ESBMC com Z3. Como pode ser visto, para o exemplo A, o CBMC apresentou melhores resultados, com um tempo total de 1553 segundos. No entanto, para o exemplo B, o melhor desempenho foi apresentado pelo verificador ESBMC utilizando solucionador Z3, em um tempo de 2432 segundos. Vários fatores podem influenciar no tempo de verificação desta propriedade (*e.g.*, ordem da planta e controlador, percentual de incertezas e heurísticas implementadas pelo solucionador). De modo geral, o ESBMC com Z3 apresentou melhores resultados para esta propriedade.

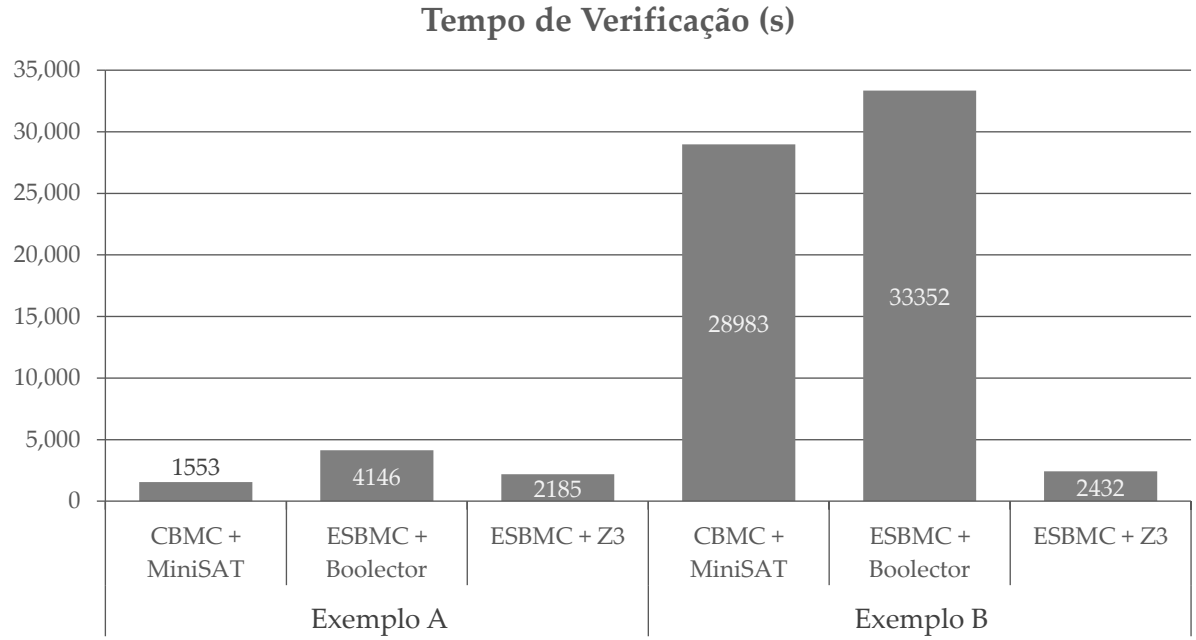

Figura 4.13: Tempo total de verificação para a propriedade de estabilidade em malha fechada considerando diferentes verificadores e solucionadores.

4.5 Resumo

Os resultados da verificação mostram que vários fatores podem influenciar o desempenho do controlador digital, dentre eles: o número de *bits* da parte inteira e fracionária definidos no formato FWL, o período de amostragem, a forma de realização, o domínio e outros que não são escopo desse estudo. O projeto ótimo e livre de falhas leva em consideração todos esses fatores e representa uma tarefa desafiadora. Além disso, a metodologia proposta pode fornecer

um apoio ao engenheiro de controle, por meio de uma ferramenta de verificação formal, capaz de encontrar diversas falhas durante a etapa de projeto.

Com base nos experimentos, pode-se perceber que mudanças nos parâmetros podem impactar no desempenho do sistema (*e.g.*, o valor Δ nas realizações delta). Os resultados mostram que alterações nos valores de delta podem prover melhores desempenhos nos arredondamentos, resultando em controladores menos frágeis (*i.e.*, com menos falhas na implementação). No entanto, reduções excessivas no valor de Δ demandam mais *bits* na parte inteira, ou *scaling* correto dos valores de entrada, resultando em uma atenção maior na ocorrência de estouros aritméticos. Com relação a ferramenta DSVerifier, nota-se que a escolha do *back-end* (*i.e.*, verificador de modelos) influencia diretamente no tempo total gasto pelas verificações. Em nossos testes, o ESBMC apresentou melhores resultados para as propriedades de controladores digitais com baixa precisão (casos de teste com 4 *bits*) e o CBMC para verificações que requerem mais *bits* de precisão. No entanto, para propriedades de malha fechada recomenda-se a utilização do verificador ESBMC com solucionador Z3.

Além disso, a metodologia proposta e implementada no DSVerifier mostrou-se útil para determinar a melhor estrutura de realização para a representação em ponto-fixo do controlador digital. Para o caso de teste 3 (um controlador digital para a planta de *ball and beam* da Quanser, ver Apêndice B), pode-se perceber facilmente que o controlador deve ser implementado em DDFII (Forma Direta Delta II) para evitar a presença de ciclo limite com entradas nulas, uma vez que esse controlador falhou nas implementações DFI, DFII e TDFII da forma direta, e também, DDFI e TDDFII da forma delta. Note que algumas falhas que aparecem no contra-exemplo (*e.g.*, ciclo limite), são difíceis de serem detectadas por ferramentas de simulação, pois só ocorrem com poucas combinações de entrada e estados nas memórias. De modo geral, a metodologia garante a corretude da implementação do controlador digital em uma profundidade k, mediante os efeitos da quantização.

Capítulo 5

Conclusões

As contribuições desse estudo podem ser divididas em três principais categorias: a introdução da verificação de modelos limitada para a validação de sistemas de controle digital; a avaliação de controladores digitais levando em consideração variáveis e parâmetros de implementação que podem afetar o seu desempenho; e uma nova metodologia para projetos mais seguros de controladores digitais.

Primeiramente, esta dissertação apresenta o potencial da verificação de modelos limitada com suporte à solucionadores de satisfação booleana e teorias do módulo da satisfatibilidade para verificar e validar controladores digitais. Nesse estudo foi utilizada a ferramenta DSVerifier usando como *back-end* diferentes verificadores de modelos considerados estado da arte (*e.g.*, CBMC e ESBMC). Tal ferramenta permite a verificação de estouros aritméticos, ciclo limite, restrições temporais, estabilidade e fase mínima para controladores realizados na Forma Direta I, Forma Direta II e Forma Direta II Transposta, incluindo seus equivalentes no domínio *delta*. O DSVerifier mostrou-se capaz de prover resultados conclusivos sobre falhas no sistema. De modo geral, a metodologia foi capaz de apresentar resultados conclusivos em 89% das verificações. Assim, o uso do DSVerifier representa uma alternativa (automática e) confiável se comparada às tradicionais ferramentas de simulação.

Em segundo lugar, um número extensivo de casos de teste permite uma análise comparativa entre realizações na formas diretas e formas delta para a implementação de controladores digitais, usando a verificação de modelos limitada para checar propriedades que são difíceis de serem verificadas em ferramentas de simulação. Os resultados experimentais mostram que as realizações na forma delta apresentam um desempenho superior quando comparado às realiza-

ções na forma direta, reduzindo a ocorrência de erros provenientes da palavra finita, preservando as propriedades de estabilidade e fase mínima, que são importantes indicadores de um sistema não frágil. No entanto, o melhor desempenho da forma delta, demanda um número maior de *bits* na parte inteira, afim de evitar estouros aritméticos durante a realização.

Por último, esse estudo fornece uma metodologia simples e interativa para projeto e verificação de controladores digitais suportadas pelo DSVerifier. Com a metodologia proposta, um projetista pode verificar, durante a fase de projeto, se o controlador digital irá apresentar o comportamento esperado, levando em consideração a sua arquitetura com limitação de recursos. Note que essa metodologia não remove todos os erros de projeto, uma vez que a verificação é executada para um número limitado de amostras, porém, é capaz de diminuí-los substancialmente.

5.1 Trabalhos Futuros

A metodologia e respectiva ferramenta foram apresentadas apenas para verificação de controladores digitais, porém, no futuro, o seu uso deve ser expandido para qualquer sistema digital. Além disso, outras formas de representações (*e.g.*, espaço de estados) devem ser exploradas, visando aumentar sua aplicabilidade.

Outro ponto a ser explorado é a detecção de falhas por meio das propriedes já suportadas pela ferramenta com objetivo de determinar a melhor implementação e realização do controlador digital, dado à limitação de recursos.

Com relação à verificação de modelos, pode-se explorar outras abordagens da técnica BMC, como por exemplo, o uso de indução k, a aplicação de invariantes e refinamento abstrato de contraexemplos, visando encontrar falhas em sistemas digitais.

Referências Bibliográficas

[1] ISTEPANIAN, R.; WHIDBORNE, J. *Digital Controller Implementation and Fragility: A Modern Perspective*. 1. ed. : Springer London, 2001. (Advances in Industrial Control). ISBN 9781852333904.

[2] YANG, G.; GUO, X.; CHE, W.; GUAN, W. *Linear Systems: Non-Fragile Control and Filtering*. 1. ed. : Taylor & Francis, 2013.

[3] MASTEN, M.; PANAHI, I. Digital Signal Processors for Modern Control Systems. In: *Control Engineering Practice*. 1997. v. 5, n. 4, p. 449–458. ISSN 0967-0661.

[4] MONMASSON, E.; CIRSTEA, M. FPGA Design Methodology for Industrial Control Systems 2014; A Review. In: *IEEE Transactions on Industrial Electronics*. 2007. v. 54, n. 4, p. 1824–1842. ISSN 0278-0046.

[5] MANTEY, P. Eigenvalue Sensitivity and State-Variable Selection. In: *IEEE Transactions on Automatic Control*. 1968. v. 13, n. 3, p. 263–269. ISSN 0018-9286.

[6] MIDDLETON, R. H.; GOODWIN, G. C. Improved Finite Word Length Characteristics in Digital Control using Delta Operators. In: *IEEE Transactions on Automatic Control*. 1986. v. 31, n. 11, p. 1015–1021. ISSN 0018-9286.

[7] WU, J.; CHEN, S.; CHU, J. Computing a FWL Stability Measure for Second Order Digital Systems. In: *8th ICARCV - Control, Automation, Robotics and Vision Conference*. 2004. v. 3, p. 1593–1598.

[8] LI, G. On Pole and Zero Sensitivity of Linear Systems. In: *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*. 1997. v. 44, n. 7, p. 583–590. ISSN 1057-7122.

[9] LI, G. On the Structure of Digital Controllers with Finite Word Length Consideration. In: *IEEE Transactions on Automatic Control*. 1998. v. 43, n. 5, p. 689–693. ISSN 0018-9286.

[10] HARNEFORS, L. Implementation of Resonant Controllers and Filters in Fixed-Point Arithmetic. In: *IEEE Transactions on Industrial Electronics*. 2009. v. 56, n. 4, p. 1273–1281. ISSN 0278-0046.

[11] ISTEPANIAN, R.; WHIDBORNE, J. Multi-Objective Design of Finite Word-Length Controller Structures. In: *Congress on Evolutionary Computation*. 1999. v. 1, p. 68.

[12] CARLETTA, J.; VEILLETTE, R.; KRACH, F.; FANG, Z. Determining Appropriate Precisions for Signals in Fixed-point IIR Filters. In: *40th Annual Design Automation Conference*. 2003. p. 656–661. ISBN 1-58113-688-9.

[13] SUNG, W.; KUM, K. Simulation-Based Word-Length Optimization Method For Fixed-Point Digital Signal Processing Systems. In: *IEEE Transactions on Signal Processing*. 1995. v. 43, n. 12, p. 3087–3090.

[14] MOHTA, V. *Finite Worldlength Effects in Fixed-point Implementations of Linear Systems*. : Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 1998.

[15] ISTEPANIAN, R.; CHEN, S. Stability issues of finite-precision controller structures for sampled-data systems. In: *International Journal of Control*. 1999. v. 72, p. 1331–1342.

[16] WO, S.; ZOU, Y.; CHEN, Q.; XU, S. Non-Fragile Controller Design for Discrete Descriptor Systems. v. 346, n. 9, p. 914 – 922, 2009. ISSN 0016-0032.

[17] BEHRMANN, G.; DAVID, A.; LARSEN, K. A Tutorial on UPPAAL. In: *International School on Formal Methods for the Design of Computer, Communication, and Software Systems*. : Springer Verlag, 2004. v. 3185, p. 200–237.

[18] LI, G. Checking Timed Buechi Automata Emptiness Efficiently. In: *Formal Methods in System Design*. 2005. v. 26, n. 3, p. 267–292. ISSN 0925-9856.

[19] JENSEN, K.; KRISTENSEN, L. *Coloured Petri Nets - Modelling and Validation of Concurrent Systems*. : Springer, 2009.

[20] MAGELLAN. Hybrid RTL Formal Verification. `http://goo.gl/uAd2Ff` - Acessado em 20 Agosto de 2015.

[21] NULL, L.; LOBUR, J. *Princípios Básicos de Arquitetura e Organização de Computadores*. 2. ed. : Bookman Companhia, 2010. ISBN 9788577807376.

[22] DAVIS, T.; SIGMON, K. *MATLAB Primer*. 7. ed. : CRC Press, 2005.

[23] JOHNSON, G. *LabVIEW Graphical Programming: Practical Applications in Instrumentation and Control*. 2. ed. : McGraw-Hill School Education Group, 1997. ISBN 007032915X.

[24] LAM, W. *Hardware Design Verification: Simulation and Formal Method-Based Approaches*. 1. ed. : Prentice Hall PTR, 2005. ISBN 0131433474.

[25] ANTA, A.; MAJUMDAR, R.; SAHA, I.; TABUADA, P. Automatic Verification of Control System Implementations. In: *10th ACM International Conference on Embedded Software*. 2010. p. 9–18. ISBN 978-1-60558-904-6.

[26] LUENGO, D.; OSES, D.; MARTINO, L. Monte Carlo Limit Cycle Characterization. In: *International Conference on Acoustics, Speech and Signal Processing*. 2014. p. 8043–8047.

[27] PATRA, A.; MUKHOPADHYAY, S. A Software Tool for Performance Evaluation of Digital Control Algorithms on Finite Wordlength Processors. *Computers & Electrical Engineering*, v. 22, n. 6, p. 403–419, 1996. ISSN 0045-7906.

[28] BARLEANU, A.; BAITOIU, V. Digital filter Optimization for C Language. In: *Advances in Electrical and Computer Engineering*. 2011. v. 11, n. 3, p. 111–114. ISSN 1582-7445.

[29] LOPEZ, J.; CAFFARENA, G.; CARRERAS, C.; NIETO-TALADRIZ, O. Fast and Accurate Computation of the Roundoff Noise of Linear Time-Invariant Systems. *Circuits, Devices Systems*, v. 2, n. 4, p. 393–408, 2008. ISSN 1751-858X.

[30] HILAIRE, T.; MENARD, D.; SENTIEYS, O. Bit Accurate Roundoff Noise Analysis of Fixed-Point Linear Controllers. In: *IEEE International Conference on Computer-Aided Control Systems*. 2008. p. 607–612.

[31] CAFFARENA, G.; CARRERAS, C.; LOPEZ, J.; FERNANDEZ, A. Fast Fixed-Point Optimization of DSP Algorithms. In: *18th VLSI System on Chip Conference*. 2010. p. 195–200.

[32] NAUD, J.; MEUNIER, Q.; MENARD, D.; SENTIEYS, O. Fixed-Point Accuracy Evaluation in the Context of Conditional Structures. In: *19th European Signal Processing Conference*. 2011. p. 1934–1938. ISSN 2076-1465.

[33] DUTERTRE, B.; RUSHBY, J.; TIWARI, A.; MUNOZ, C.; SIMINICEANU, R. Verification and Automated Testing for Diagnostic and Monitoring Systems. *AIAA Guidance, Navigation and Control Conference and Exhibit, American Institute of Aeronautics and Astronautics*, n. 2008-6802, 2008. ISSN 10877215.

[34] SIMKO; GABOR; JACKSON; ETHAN, K. A Bounded Model Checking Tool for Periodic Sample-hold Systems. In: *17th International Conference on Hybrid Systems: Computation and Control*. 2014. p. 157–162. ISBN 978-1-4503-2732-9.

[35] PRABHU, S.; DASGUPTA, P. Model Checking Controllers with Predicate Inputs. In: *12th International Conference on Embedded Systems*. 2013. p. 332–337. ISSN 1063-9667.

[36] COX, A.; SANKARANARAYANAN, S.; CHANG, B. A Bit Too Precise? Bounded Verification of Quantized Digital Filters. In: *18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. 2012. v. 7214, p. 33–47. ISBN 978-3-642-28755-8.

[37] COX, A.; SANKARANARAYANAN, S.; CHANG, B. A Bit Too Precise? Verification of Quantized Digital Filters. In: *International Journal on Software Tools for Technology Transfer*. 2014. v. 16, n. 2, p. 175–190.

[38] ISMAIL, H.; BESSA, I.; CORDEIRO, L.; FILHO, E.; CHAVES, J. DSVerifier: A Bounded Model Checking Tool for Digital Systems. In: *Model Checking Software*. : Springer, 2015. v. 9232, p. 126–131. ISBN 978-3-319-23403-8.

[39] ABREU, R.; CORDEIRO, L.; FILHO, E. Verifying Fixed-Point Digital Filters using SMT-Based Bounded Model Checking. *XXXI Simposio Brasileiro de Telecomunicacoes*, 2013.

[40] BESSA, I.; ABREU, R.; CORDEIRO, L.; FILHO, J. E. SMT-Based Bounded Model Checking of Fixed-Point Digital Controllers. *41st Annual Conference of the IEEE Industrial Electronics Society*, p. 295–301, 2014.

[41] FADALI, M.; VISIOLI, A. Analysis and Design. In: *Digital Control Engineering*. : Academic Press, 2009. ISBN 978-0-12-374498-2.

[42] OPPENHEIM, A.; WILLSKY, A.; NAWAB, S. *Signals & Systems*. 2. ed. : Prentice-Hall, Inc., 1996. ISBN 0-13-814757-4.

[43] PROAKIS, J.; MANOLAKIS, D. *Digital Signal Processing: Principles, Algorithms, and Applications*. 3. ed. : Prentice-Hall, 1996. ISBN 0-13-373762-4.

[44] DINIZ, P.; NETTO, S.; SILVA, E. *Digital Signal Processing: System Analysis and Design*. 2. ed. : Cambridge University Press, 2002. ISBN 0521781752.

[45] MIDDLETON, R.; GOODWIN, G. Improved Finite Word Length Characteristics in Digital Control using Delta Operators. *IEEE Transactions on Automatic Control*, v. 31, n. 11, p. 1015–1021, 1986. ISSN 0018-9286.

[46] ISTEPANIAN, R.; WHIDBORNE, J. *Digital Controller Implementation and Fragility: A Modern Perspective*. 1. ed. : Advances in Industrial Control, Springer, London, 2001.

[47] PERETZ, M.; BEN-YAAKOV, S. Digital Control of Resonant Converters: Resolution Effects on Limit Cycles. *IEEE Transactions on Power Electronics*, v. 25, n. 6, p. 1652–1661, 2010. ISSN 0885-8993.

[48] GRANAS, A.; DUGUNDJI, J. *Fixed Point Theory*. 1. ed. : Springer, 2003. (Springer Monographs in Mathematics). ISBN 978-0-387-00173-9.

[49] JACKSON, L. *Digital Filters and Signal Processing: With MATLAB Exercises*. 3. ed. : Kluwer Academic Publishers, 1996. ISBN 079239559X.

[50] MIDDLETON; RICHARD, H.; GOODWIN, C. *Digital Control and Estimation: A Unified Approach*. 1. ed. : Prentice Hall Professional Technical Reference, 1990. ISBN 0132116650.

[51] GEVERS; MICHEL; LI, G. *Parametrizations in Control, Estimation, and Filtering Problems: Accuracy Aspects*. 1. ed. : Springer-Verlag New York, 1993. ISBN 0387198210.

[52] KEEL, L.; BHATTACHARYYA, S. Robust, Fragile, or Optimal? *IEEE Transactions on Automatic Control*, v. 42, n. 8, p. 1098–1105, 1997. ISSN 0018-9286.

[53] JACKSON, L.; KAISER, J.; MCDONALD, H. An approach to the Implementation of Digital Filters. *IEEE Transactions on Audio and Electroacoustics*, v. 16, n. 3, p. 413–421, 1968. ISSN 0018-9278.

[54] GANG, L.; ANDERSON, B.; GEVERS, M.; PERKINS, J. Optimal FWL Design of State-Space Digital Systems with Weighted Sensitivity Minimization and Sparseness Consideration. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, v. 39, n. 5, p. 365–377, 1992. ISSN 1057-7122.

[55] LOPEZ, B.; HILAIRE, T.; DIDIER, L. Formatting Bits to Better Implement Signal Processing Algorithms. *4th International Conference on Pervasive and Embedded Computing and Communication Systems*, p. 104–111, 2014.

[56] MENARD, D.; ROCHER, R.; SENTIEYS, O.; SIMON, N.; DIDIER, L.; HILAIRE, T.; LOPEZ, B.; GOUBAULT, E.; PUTOT, S.; VEDRINE, F.; NAJAHI, A.; REVY, G.; FANGAIN, L.; SAMOYEAU, C.; LEMONNIER, F.; CLIENTI, C. Design of Fixed-Point Embedded Systems (DEFIS) French ANR Project. In: *Conference on Design and Architectures for Signal and Image Processing*. 2012. p. 1–2.

[57] PARASHAR, K.; MENARD, D.; SENTIEYS, O. A Polynomial Time Algorithm for Solving the Word-Length Optimization Problem. In: *International Conference on Computer-Aided Design*. 2013. p. 638–645. ISSN 1092-3152.

[58] SARBISHEI, O.; RADECKA, K.; ZILIC, Z. Analytical Optimization of Bit-Widths in Fixed-Point LTI Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 31, n. 3, p. 343–355, 2012. ISSN 0278-0070.

[59] MENARD, D.; ROCHER, R.; SENTIEYS, O. Analytical fixed-point accuracy evaluation in linear time-invariant systems. *IEEE Transactions on Circuits and Systems I: Regular Papers*, v. 55, n. 10, p. 3197–3208, 2008. ISSN 1549-8328.

[60] SKAF, J.; BOYD, S. Filter Design with Low Complexity Coefficients. *vIEEE Transactions on Signal Processing*, v. 56, n. 7, p. 3162–3169, 2008.

[61] ISTEPANIAN, R.; WHIDBORNE, J. Multi-Objective Design of Finite Word-Length Controller Structures. In: *Congress on Evolutionary Computation*. 1999. v. 1.

[62] BALAKRISHNAN, V.; BOYD, S. On Computing the Worst-Case Peak Gain of Linear Systems. In: *31st IEEE Conference on Decision and Control*. 1992. p. 2191–2192 vol.2.

[63] VIASSOLO, D. E. Realizations for Fixed-Point Implementation of Controllers and Filters with Peak-to-Peak Scaling. In: *American Control Conference*. 2003. v. 1, p. 83–88. ISSN 0743-1619.

[64] PETERCHEV, A.; SANDERS, S. Quantization Resolution and Limit Cycling in Digitally Controlled PWM Converters. In: *32nd Annual Power Electronics Specialists Conference*. 2001. v. 2, p. 465–471 vol.2. ISSN 0275-9306.

[65] BAUER, P. H.; LECLERC, L.-J. A Computer-Aided Test for the Absence of Limit Cycles in Fixed-Point Digital Filters. *IEEE Transactions on Signal Processing*, v. 39, n. 11, p. 2400–2410, 1991. ISSN 1053-587X.

[66] UTRILLA-MANSO, M.; LOPEZ-FERRERAS, F.; CAMPO, D.; MARTIN-MARTIN, P. A Computer-Aided Test for the Characterization of Parasitic Oscillations in IIR Digital Filters. In: *2nd International Symposium on Image and Signal Processing and Analysis*. 2001. p. 475–478.

[67] DJEBBARI, A.; BELBACHIR, M. F.; ROUVAEN, J. M. A Fast Exhaustive Search Algorithm for Checking Limit Cycles in Fixed-Point Digital Filters. *Signal Process*, Elsevier, v. 69, n. 2, p. 199–205, 1998. ISSN 0165-1684.

[68] MARTINEZ, R.; FERRERAS, F.; CAMPO, D. D.; MANSO, M. An Alternative to Exhaustive Search Scheme for Limit Cycles Behavior in Digital Filters. WSEAS ISPRA, p. 1221–1226, 2002.

[69] PREMARATNE, K.; KULASEKERE, E.; BAUER, P. H.; LECLERC, L.-J. An Exhaustive Search Algorithm for Checking Limit Cycle Behavior of Digital Filters. *IEEE Transactions on Signal Processing*, v. 44, n. 10, p. 2405–2412, 1996. ISSN 1053-587X.

[70] CAMPO, J. D. . O.; CRUZ-ROLDAN, F.; UTRILLA-MANSO, M. Tighter Limit Cycle Bounds for Digital Filters. *IEEE Signal Processing Letters*, v. 13, n. 3, p. 149–152, 2006. ISSN 1070-9908.

[71] GREEN, B.; TURNER, L. New Limit Cycle Bounds for Digital Filters. *IEEE Transactions on Circuits and Systems*, v. 35, n. 4, p. 365–374, 1988. ISSN 0098-4094.

[72] OGATA, K. *Engenharia de controle moderno*. 4. ed. : Pearson Prentice Hall, 1982. ISBN 9788587918239.

[73] MOURA, L.; BJORNER, N. Satisfiability Modulo Theories: An Appetizer. In: *Formal Methods: Foundations and Applications*. : Springer-Verlag, 2009. p. 23–36. ISBN 978-3-642-10451-0.

[74] MOURA, L.; BJORNER, N. Z3: An Efficient SMT Solver. In: *14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. : Springer-Verlag, 2008. p. 337–340. ISBN 3-540-78799-2, 978-3-540-78799-0.

[75] BRUMMAYER, R.; BIERE, A. Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays. In: *Tools and Algorithms for the Construction and Analysis of Systems*. : TACAS, 2009. v. 5505, p. 174–177.

[76] BJORNER, N.; MOURA, L. *Z310: Applications, Enablers, Challenges and Directions*.

[77] LEE, E. Cyber Physical Systems: Design Challenges. In: *11th IEEE Symposium on Object Oriented Real-Time Distributed Computing*. : IEEE Computer Society, 2008. p. 363–369. ISBN 978-0-7695-3132-8.

[78] BEYER, D. Status Report on Software Verification. In: *Tools and Algorithms for the Construction and Analysis of Systems*. : Springer Berlin Heidelberg, 2014. v. 8413, p. 373–388.

[79] BIERE, A.; CIMATTI, A.; CLARKE, E.; STRICHMAN, O.; ZHU, Y. Bounded Model Checking. In: *Advances in Computers*. : IOS Press, 2003. v. 58, p. 457–481.

[80] BARRETT, C.; SEBASTIANI, R.; TINELLI, C. Satisfiability Modulo Theories. In: *Handbook of Satisfiability*. : IOS Press, 2009. v. 185, p. 825–885.

[81] CORDEIRO, L.; FISCHER, B.; MARQUES, J. SMT-Based Bounded Model Checking for Embedded ANSI-C Software. In: *IEEE/ACM International Conference on Automated Software Engineering*. : IEEE Computer Society, 2009. p. 137–148. ISBN 978-0-7695-3891-4.

[82] CORDEIRO, L. C.; FISCHER, B. Verifying Multi-Threaded Software using SMT-Based Context-Bounded Model Checking. In: *33rd International Conference on Software Engineering*. 2011. p. 331–340.

[83] KROENING, D.; TAUTSCHNIG, M. CBMC – C Bounded Model Checker. In: *Tools and Algorithms for the Construction and Analysis of Systems*. : Springer Berlin Heidelberg, 2014. v. 8413, p. 389–391. ISBN 978-3-642-54861-1.

[84] FALKE, S.; MERZ, F.; SINZ, C. LLBMC: Improved Bounded Model Checking of C Programs using LLVM. In: *Tools and Algorithms for the Construction and Analysis of Systems*. : Springer Berlin Heidelberg, 2013. v. 7795, p. 623–626. ISBN 978-3-642-36741-0.

[85] BESSA, I.; IBRAHIM, H.; CORDEIRO, L.; FILHO, J. E. Verification of Delta Form Realization in Fixed-Point Digital Controllers Using Bounded Model Checking. *Brazilian Symposium on Computing Systems Engineering*, p. 49–54, 2014.

[86] BAIER, C.; KATOEN, J. *Principles of Model Checking*. 1. ed. : The MIT Press, 2008. ISBN 026202649X.

[87] JHALA, R.; MAJUMDAR, R. Software Model Checking. *ACM Computing Surveys*, ACM, v. 41, n. 4, p. 21:1–21:54, 2009. ISSN 0360-0300.

[88] CLARKE, E.; KROENING, D.; LERDA, F. A Tool for Checking ANSI-C Programs. In: *Tools and Algorithms for the Construction and Analysis of Systems*. : Springer, 2004. v. 2988, p. 168–176. ISBN 978-3-540-21299-7.

[89] EEN, N.; SORENSSON, N. An Extensible SAT-solver. In: *Theory and Applications of Satisfiability Testing*. : Springer Berlin Heidelberg, 2004. v. 2919, p. 502–518. ISBN 978-3-540-20851-8.

[90] CORDEIRO, L. C.; FISCHER, B.; MARQUES-SILVA, J. SMT-Based Bounded Model Checking for Embedded ANSI-C Software. *IEEE Transactions on Software Engineering*, v. 38, n. 4, p. 957–974, 2012.

[91] CORDEIRO, L. C.; FISCHER, B.; MARQUES-SILVA, J. SMT-Based Bounded Model Checking for Embedded ANSI-C Software. In: *24th IEEE/ACM International Conference on Automated Software Engineering*. 2009. p. 137–148. ISSN 1938-4300.

[92] MORSE, J.; CORDEIRO, L. C.; NICOLE, D.; FISCHER, B. Model Checking LTL Properties over ANSI-C Programs with Bounded Traces. *Software and System Modeling*, v. 14, n. 1, p. 65–81, 2015.

[93] MORSE, J.; CORDEIRO, L. C.; NICOLE, D.; FISCHER, B. Context-Bounded Model Checking of LTL Properties for ANSI-C Software. In: *9th International Conference of Software Engineering and Formal Methods*. 2011. v. 7041, p. 302–317.

[94] ROCHA, H.; BARRETO, R. S.; CORDEIRO, L. C.; NETO, A. D. Understanding Programming Bugs in ANSI-C Software Using Bounded Model Checking Counter-Examples. In: *9th International Conference of Integrated Formal Methods*. 2012. v. 7321, p. 128–142.

[95] TRINDADE, A.; CORDEIRO, L. Applying SMT-based Verification to Hardware/-Software Partitioning in Embedded Systems. *Design Automation for Embedded Systems*, Springer US, p. 1–19, 2015. ISSN 0929-5585.

[96] TRINDADE, A.; ISMAIL, H.; CORDEIRO, L. C. Applying Multi-Core Model Checking to Hardware-Software Partitioning in Embedded Systems. *Brazilian Symposium on Computing Systems Engineering*, abs/1509.02492, 2015.

[97] ROCHA, H.; ISMAIL, H.; CORDEIRO, L.; BARRETO, R. Model Checking C Programs with Loops via k-induction and Invariants. *Report Paper*, 2015.

[98] ROCHA, H.; ISMAIL, H.; CORDEIRO, L. C.; BARRETO, R. S.

[99] PEREIRA, P.; ALBUQUERQUE, H.; MARQUES, H.; SILVA, I.; SANTOS, V.; FERREIRA, R.; CARVALHO, C.; CORDEIRO, L. Verificação de Kernels em Programas CUDA usando Bounded Model Checking. XVI Simpósio em Sistemas Computacionais de Alto Desempenho, 2015.

[100] CORDEIRO, L.; MORSE, J.; NICOLE, D.; FISCHER, B. Context-Bounded Model Checking with ESBMC 1.17. In: *Tools and Algorithms for the Construction and Analysis of Systems*. : Springer, 2012. v. 7214, p. 534–537.

[101] MORSE, J.; NICOLE, L. C. D.; FISCHER, B. Handling Unbounded Loops with ESBMC 1.20. In: *Tools and Algorithms for the Construction and Analysis of Systems*. : Springer Berlin Heidelberg, 2013. p. 619–622. ISBN 978-3-642-36741-0.

[102] MORSE, J.; RAMALHO, M.; CORDEIRO, L.; NICOLE, D.; FISCHER, B. ESBMC 1.22. In: *Tools and Algorithms for the Construction and Analysis of Systems*. : Springer Berlin Heidelberg, 2014. p. 405–407. ISBN 978-3-642-54861-1.

[103] KEEL, L.; BHATTACHARYYA, S. Stability Margins and Digital Implementation of Controllers. In: *American Control Conference*. 1998. v. 5, p. 2852–2856 vol.5. ISSN 0743-1619.

[104] OGATA, K. *Discrete-time Control Systems*. 2. ed. : Prentice-Hall, 1987. ISBN 0-132-16102-8.

[105] BURNS, A.; WELLINGS, A. *Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX*. 4. ed. : Addison-Wesley Educational Publishers Inc, 2009. ISBN 0321417453.

[106] KIM, S.; PATEL, H.; EDWARDS, S. Using a Model Checker to Determine Worst-case Execution Time. *Computer Science Technical Report*, Columbia University, 2009.

[107] PATTERSON, D.; HENNESSY, L. *Organização e Projeto de Computadores: A Interface Hardware/Software*. 3. ed. : CAMPUS - RJ, 2005. ISBN 9788535215212.

[108] ASTROM, K.; WITTENMARK, B. *Computer-controlled Systems*. 3. ed. : Prentice-Hall, 1997. ISBN 0-13-314899-8.

[109] SIMPSON, H. Eigen: a C++ Linear Algebra Library. `http://eigen.tuxfamily.org` - Acessado em 21 de agosto de 2015.

[110] FEUER; ARIE; GOODWIN, G. *Sampling in Digital Signal Processing and Control*. 1. ed. : Birkhauser Boston, 1996. ISBN 0-8176-3934-9.

Apêndice A

Digital-Systems Verifier (DSVerifier)

O DSVerifier (*Digital-Systems Verifier*) inicialmente foi criado como um módulo interno para o ESBMC (*Efficient SMT-based Context-Bounded Model Checker*) [81], desenvolvido com objetivo de adicionar o suporte à verificação de sistemas digitais. Atualmente, a ferramenta também possui suporte ao CBMC (*C Bounded Model Checker*) [88] e pode ser conectada a outros verificadores de modelos limitados.

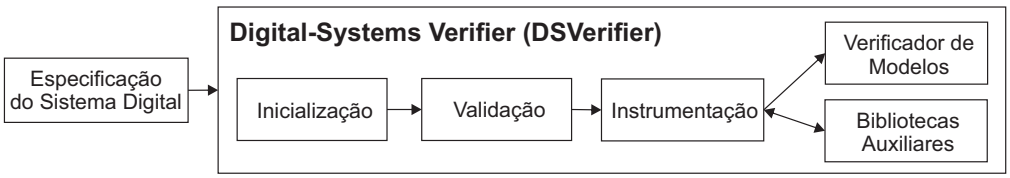

Figura A.1: Visão geral da arquitetura da ferramenta de verificação.

A Figura A.1 apresenta uma visão geral da arquitetura da ferramenta de verificação de sistemas digitais. Em geral, ela fornece funções relacionadas à quantização em aritmética de ponto-fixo e diferentes formas de realizações. O DSVerifier executa três procedimentos principais, que são: inicialização, validação e instrumentação. Após receber a especificação de um sistema digital, a ferramenta inicializa seus parâmetros internos para a quantização, *i.e.*, gera os números máximo e mínimo permitidos pela representação da palavra finita e o fator de escala. Durante o processo de validação, o DSVerifier checa se todos os parâmetros necessários para a verificação foram informados pelo usuário. Por fim, a ferramenta adiciona chamadas explícitas para a máquina de verificação, com funções disponíveis no verificador de modelos a ser utilizado, como por exemplo, no ESBMC (*e.g.*, *__ESBMC_assume* e *__ESBMC_assert*) e no CBMC (*e.g.*, *__CPROVER_assume* e *__CPROVER_assert*), no intuito de checar pos-

síveis violações nas propriedades. Tal procedimento, caso necessário, pode utilizar recursos de bibliotecas externas ao DSVerifier, como por exemplo a extração de autovalores por meio da *Eigen* [109]. Caso a ferramenta de verificação encontre uma violação, é produzido um contra-exemplo. Caso contrário, o projeto avaliado está pronto para ser embarcado no sistema de computação proposto.

A.1 Uso do DSVerifier

A.1.1 Verificação de Controladores Digitais

Para exemplificar a metodologia de verificação de controladores digitais, será utilizado um controlador de segunda ordem (número 11), que foi retirado dos nossos casos de teste (ver Apêndice B). De modo que, sua representação é dada por:

$$C(z) = \frac{1.611z^2 + 3.079z - 3.794}{z^2 + 1.084z + 0.1289} \tag{A.1}$$

```
1  #include <dsverifier.h>
2  digital_system ds = {
3      .b = { 1.611, 3.079, -3.794 },
4      .b_size = 3,
5      .a = { 1.0, 1.084, 0.1289 },
6      .a_size = 3,
7      .sample_time = 0.4
8  };
9  implementation impl = {
10     .int_bits = 11,
11     .frac_bits = 4,
12     .min = -5.0,
13     .max = 5.0
14 };
15 hardware hw = {
16     .clock = 50000000,
17     .assembly = {
18         .push = 1,  .in = 1,   .sbiw = 1, .cli = 1, .out = 1,  .std = 1,
19         .ldd = 1,   .subi = 1, .sbci = 1, .lsl = 1, .rol = 1,  .add = 1,
20         .adc = 1,   .adiw = 1, .rjmp = 1, .mov = 1, .sbc = 1,  .ld = 1,
21         .rcall = 1, .cp = 1,   .cpc = 1,  .ldi = 1, .brge = 1, .pop = 1,
22         .ret = 1,   .st = 1,   .brlt = 1, .cpi = 1
23     }
24 };
```

Figura A.2: Arquivo de entrada para a verificação de um controlador digital no DSVerifier, contendo especificações do *hardware*.

O usuário utiliza como entrada um arquivo ANSI-C, que segue o padrão descrito na Figura A.2. Tal arquivo é composto pela especificação do sistema digital (ds) com numerador (ds.b = {1.611, 3.079, −3.794}), denominador (ds.a = {1.0, 1.084, 0.1289}) e período de amostragem (ds.sample_time = 0.4). Nele também são fornecidos dados a respeito da sua implementação (impl), que contém, o número de *bits* da parte inteira (impl.int_bits = 11), a precisão (impl.frac_bits = 4), *i.e.*, número de *bits* da parte fracionária, e o intervalo dinâmico de entradas (impl.min = −5.0 e impl.max = 5.0). Além disso, é possível incluir informações sobre a plataforma de *hardware* onde o controlador digital será embarcado, contendo o *clock* do processador, e a quantidade de ciclos para cada instrução *assembly* utilizada durante a realização. Em geral, esses dados são necessários para a verificação de restrições temporais. A chamada da ferramenta para tal arquivo é demonstrada na Seção A.1.3.

A.1.2 Verificação de Controladores Digitais em Malha Fechada

Para exemplificar a metodologia de verificação de controladores digitais e plantas em malha fechada, serão utilizados uma planta $G(s)$ e um controlador $C(s)$, extraídos do exemplo A [103], ambos discretizados com período de amostragem de 0.05 segundos:

$$G(z) = \frac{0.0500422033454653z - 0.0526068264456340}{z^2 - 2.05640034257636z + 1.05127109637602} \quad C(z) = \frac{18.53046z - 16.7960}{z - 2.085356} \tag{A.2}$$

Para essa metodologia, o usuário informa um arquivo ANSI-C que segue o padrão visto na Figura A.3. Tal arquivo contém duas especificações de sistemas de digitais, a primeira para a planta discretizada e a segunda para o controlador digital, de modo que, cada especificação contém seus respectivos numeradores e denominadores. Além disso, é possível adicionar um percentual de incertezas para cada coeficiente da planta, chamados de *plant.b_uncertainty* e *plant.a_uncertainty*. O exemplo mostrado na Figura A.3, descreve um modelo da planta considerando variações de $\pm 0.25\%$ em todos os coeficientes do numerador e denominador. Atualmente, para controle em malha fechada, somente a propriedade de estabilidade considerando incertezas paramétricas é suportada pelo DSVerifier. Em especial, essa propriedade não utiliza uma profundidade k, pois é baseada apenas no valor dos coeficientes. A chamada da ferramenta para tal arquivo, é demonstrada na Seção A.1.3.

```
#include <dsverifier.h>
digital_system plant = {
    .b = { 0.0500422033454653, -0.0526068264456340 },
    .b_uncertainty = { 0.25, 0.25 },
    .b_size = 2,
    .a = { 1.0, -2.05640034257636, 1.05127109637602 },
    .a_uncertainty = { 0.25, 0.25, 0.25 },
    .a_size = 3
};
digital_system controller = {
    .b = { 18.53046, -16.7960 },
    .b_size = 2,
    .a = { 1.0, -2.085356 },
    .a_size = 2
};
implementation impl = {
    .int_bits = 16,
    .frac_bits = 12
};
```

Figura A.3: Arquivo de entrada para a verificação em malha fechada no DSVerifier.

A.1.3 Versão Linha de Comando

A ferramenta é executada utilizando o seguinte comando:

```
dsverifier <file> --realization <i> --property <j> --x-size <k>
```

onde $< file >$ corresponde ao arquivo de especificação do sistema digital, $< i >$ é a realização escolhida, $< j >$ é a propriedade a ser verificada e $< k >$ é o limite de verificação (o número de vezes que o sistema digital será desdobrado).

Atualmente 12 realizações são suportadas pelo DSVerifier, que são: forma direta I (DFI), forma direta II (DFII), forma direta II transposta (TDFII), forma direta delta I (DDFI), forma direta delta II (DDFII), forma direta delta transposta II (TDDFII), forma direta em cascata I (CDFI), forma direta em cascata II (CDFII), forma direta transposta em cascata II (CTDFII), forma direta delta em cascata I (CDDFI), forma direta delta em cascata II (CDDFII) e forma direta delta transposta em cascata II (CTDDFII). Incluindo o suporte à verificação de controladores digitais em 5 diferentes propriedades, que são: estouros aritméticos (OVERFLOW), ciclo limite considerando entradas constantes nulas (ZERO_INPUT_LIMIT_CYCLE) ou não determinísticas (LIMIT_CYCLE), restrições temporais (TIMING), estabilidade (STABILITY) e fase mínima (MINIMUM_PHASE). Além disso, é possível verificar a estabilidade em malha fechada (STABILITY_CLOSED_LOOP), considerando incertezas paramétricas na planta.

A.1.4 Versão Gráfica

Visando facilitar a verificação de sistemas digitais, a Figura A.4 mostra o ambiente gráfico (*Graphical User Interface*, GUI) que foi desenvolvido para o DSVerifier, melhorando a sua usabilidade e consequentemente podendo atrair mais projetistas e engenheiros.

A Figura A.6) mostra o do ambiente gráfico do DSVerifier, nela o usuário pode fornecer todos os parâmetros necessários para a verificação: especificação do sistema digital, informações sobre o processador e as propriedades a serem verificadas.

Outra funcionalidade interessante para o ambiente gráfico proposto é a execução em paralelo das tarefas de verificação, que pode diminuir potencialmente o tempo total de verificação gasto pelo DSVerifier. A GUI permite que o usuário tenha acesso gráfico aos resultados obtidos no contraexemplo incluindo documentação, casos de teste e publicações sobre a ferramenta (Ver Figuras A.5 e A.7).

Como requisito para a execução da GUI, é necessário que o usuário tenha em seu computador pelo menos a máquina virtual (*Java RunTime Environment*, JRE) versão 8.0, atualização 40 (jre1.8.0_40)[1], devido aos componentes do JavaFX.

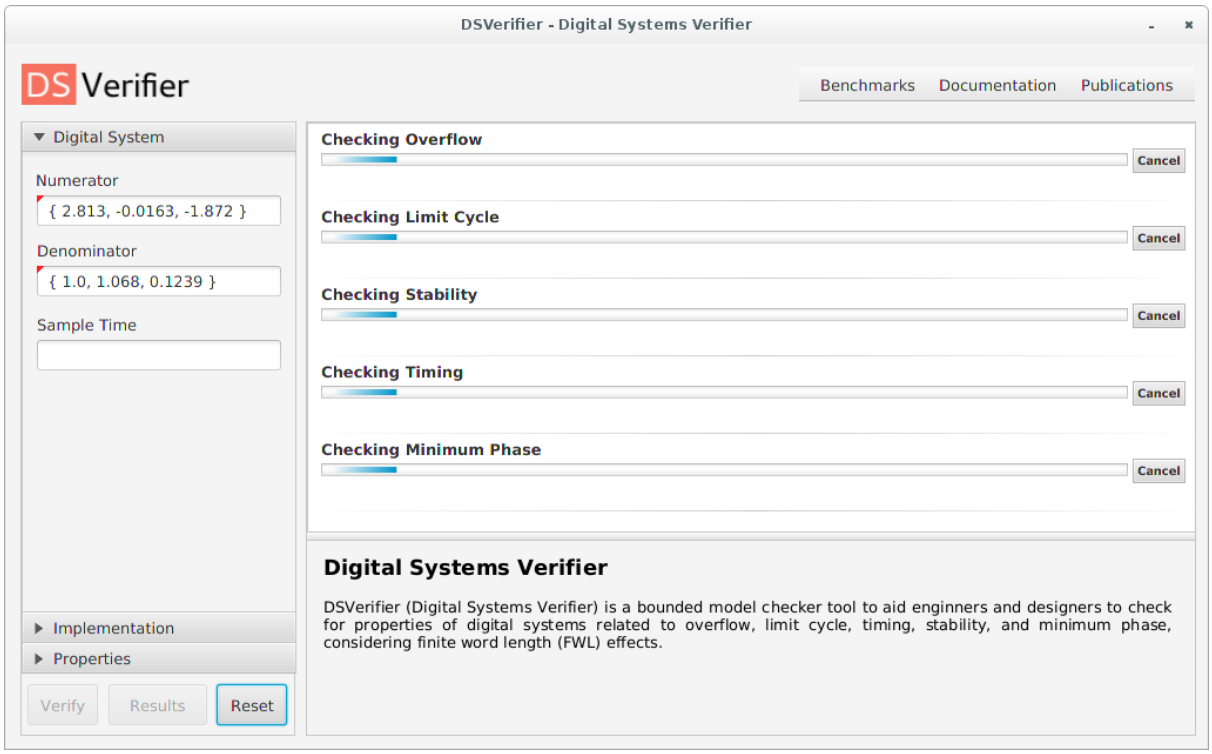

Figura A.4: Ambiente gráfico desenvolvido para o DSVerifier.

[1] http://www.oracle.com/technetwork/java/javase/8u40-relnotes-2389089.html

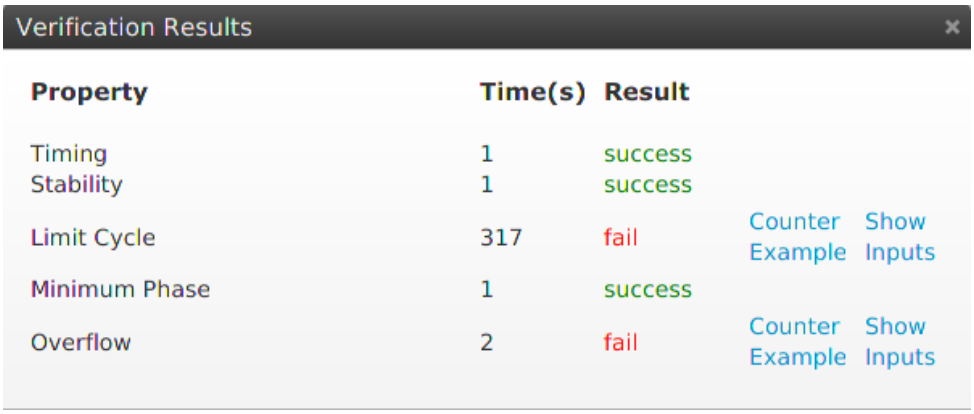

Figura A.5: Resultados exibidos pela GUI do DSVerifier após verificação do sistema digital.

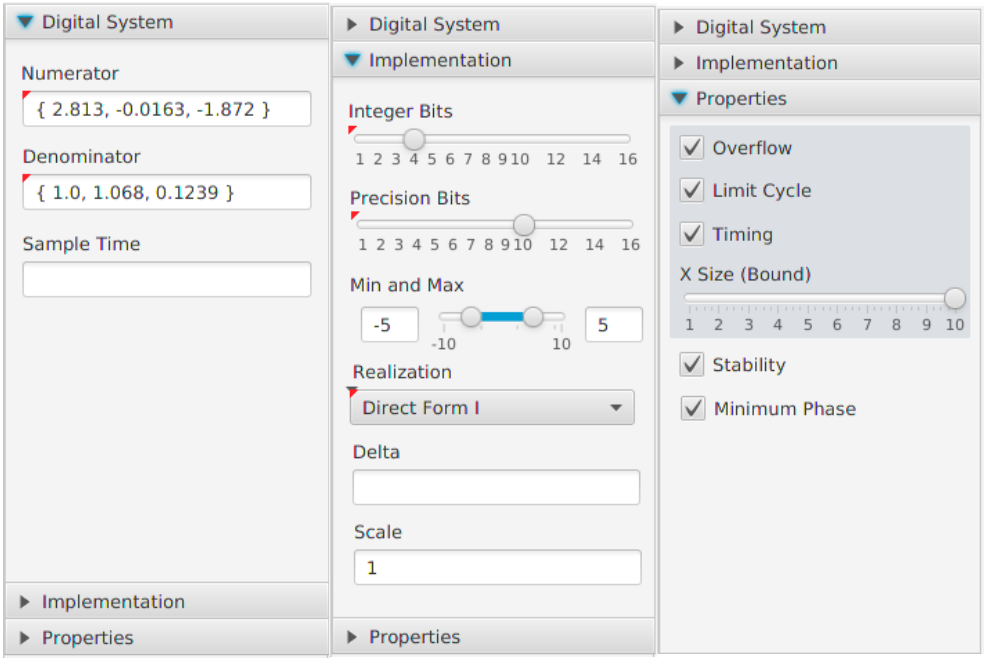

Figura A.6: Inclusão de parâmetros para verificação no ambiente gráfico do DSVerifier.

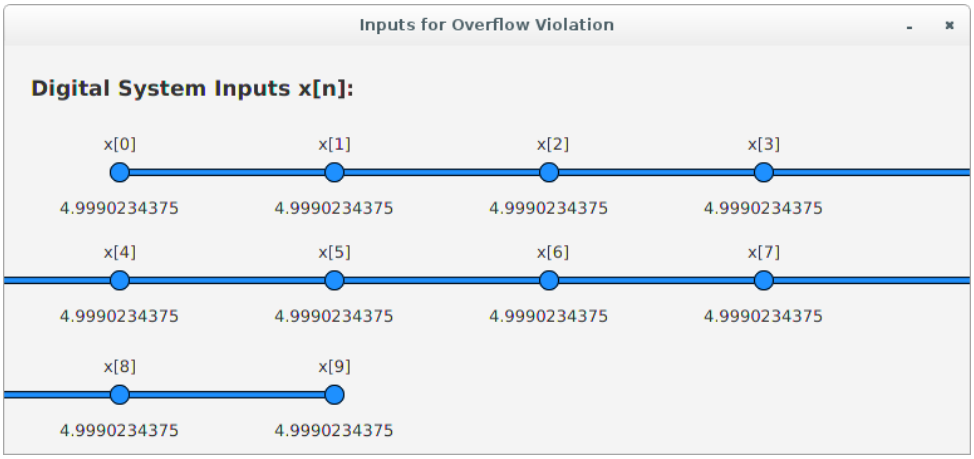

Figura A.7: Contraexemplo com as entradas utilizadas para encontrar uma violação de estouro aritmético no DSVerifier.

Apêndice B

Casos de Teste

Este apêndice expande os resultados experimentais apresentados no Capítulo 4, provendo mais detalhes sobre os controladores digitais projetados. A tabela B.1 mostra os 20 controladores digitais que foram desenvolvidos para plantas apresentadas nesse trabalho.

	Numerador	Denominador	Intervalo	*bits*
1	{ 0.1, −0.2819, 0.2637, −0.08187 }	{ 1.0, −2.574, 2.181, −0.6068 }	[−4.0,4.0]	$< 3,12 >$
2	{ 0.1, −0.28, 0.26, −0.08 }	{ 1.0, −2.57, 2.18, −0.60 }	[−5.0,5.0]	$< 4,12 >$
3	{ 0.1, −0.28, 0.26, −0.08 }	{ 1.0, −2.57, 2.18, −0.60 }	[−6.0,6.0]	$< 4,12 >$
4	{ 0.1, −0.28, 0.26, −0.08 }	{ 1.0, −2.57, 2.18, −0.60 }	[−10.0,10.0]	$< 5,12 >$
5	{ 7.936, −2.919 }	{ 1.0, 0.389 }	[−9.0,9.0]	$< 9,12 >$
6	{ 1.2670, −0.8493 }	{ 1.0, −0.2543 }	[−10.0,10.0]	$< 4,12 >$
7	{ 1.4340, −1.174 }	{ 1.0, −0.608 }	[−3.0,3.0]	$< 4,12 >$
8	{ 1.5000, −1.357 }	{ 1.0, −0.801 }	[−3.0,3.0]	$< 4,12 >$
9	{ 1.5610, −1.485 }	{ 1.0, −0.96 }	[−3.0,3.0]	$< 4,12 >$
10	{ 1.5840, −1.553 }	{ 1.0, −0.96 }	[−3.0,3.0]	$< 4,12 >$
11	{ 1.611, 3.079, −3.794 }	{ 1.0, 1.084, 0.1289 }	[−5.0,5.0]	$< 11,12 >$
12	{ −1.099, 2.978, −1.812 }	{ 1.0, 0.9068, −0.06514 }	[−4.0,4.0]	$< 11,12 >$
13	{ −0.4603, 1.006, −0.5421 }	{ 1.0, 0.5949, −0.3867 }	[−4.0,4.0]	$< 12,12 >$
14	{ −1.239, 2.565, −1.323 }	{ 1.0, 0.3423, −0.6468 }	[−4.0,4.0]	$< 12,12 >$
15	{ −2.813, 5.719, −2.905 }	{ 1.0, 0.18330, −0.8107 }	[−6.0,6.0]	$< 15,12 >$
16	{ −0.753, 1.519, −0.766 }	{ 1.0, 0.0762700, −0.9212 }	[−3.0,3.0]	$< 15,12 >$
17	{ −1.553, 3.119, −1.566 }	{ 1.0, 0.0387300, −0.96 }	[−3.0,3.0]	$< 15,12 >$
18	{ 60.0, −50.0 }	{ 1.0, 0.0 }	[−1.0,1.0]	$< 8,12 >$
19	{ 110.0, −100.0 }	{ 1.0, 0.0 }	[−1.0,1.0]	$< 9,12 >$
20	{ 135.0, −260.0, 125.0 }	{ 1.0, −1.0, 0.0 }	[−1.0,1.0]	$< 10,12 >$

Tabela B.1: Controladores digitais projetados para as plantas da Seção 4.3.1.

Apêndice C

Publicações

C.1 Referente à Pesquisa

- Bessa, I., **Ismail, H.**, V., Frutuoso, A., Cordeiro, L., Filho, E. B., Chaves Filho, J. E. *DSVerifier-Aided Verification Applied to Attitude Control Software in Unmanned Aerial Vehicles*. In 38th International Conference on Software Engineering (ICSE), 2016. **(Submetido)**

- **Ismail, H.**, Bessa, I. V., Cordeiro, L., Chaves Filho, J. E. *DSVerifier: A Bounded Model Checking Tool for Digital Systems*. In SPIN Workshop on Model Checking of Software (SPIN 2015), LNCS 9232, p. 126-131, 2015 **(Publicado)**

- Bessa, I. V., **Ismail, H.**, Cordeiro, L., Chaves Filho, J. E. *Verification of Fixed-Point Digital Controllers Using Direct and Delta Forms Realizations*. In Design Automation for Embedded Systems (DAES), 2015. **(Em Revisão)**

- Bessa, I. V., **Ibrahim, H.**, Cordeiro, L., Chaves Filho, J. E. *Verification of Delta Form Realization in Fixed-Point Digital Controllers Using Bounded Model Checking*. In IV Brazilian Symposium on Computing System Engineering (SBESC), 2014. DOI: 10.1109/SBESC.2014.14 **(Publicado)**

C.2 Contribuições em outras Pesquisas

- Gadelha, M., **Ismail, H.**, Cordeiro, L., Barreto, R. *Handling Loops in Bounded Model Checking of C Programs via k-Induction* In International Journal on Software Tools for

Technology Transfer (STTT), 2015. DOI: 10.1007/s10009-015-0407-9 **(Publicado)**

- Rocha, H., **Ismail, H.**, Cordeiro, L., Barreto, R. *Model Checking Embedded C Software using k-Induction and Invariants*. To appear in the V Brazilian Symposium on Computing System Engineering (SBESC), 2015. **(Aceito)**

- Trindade, A. **Ismail, H.**, Cordeiro, L. *Applying Multi-Core Model Checking to Hardware-Software Partitioning in Embedded Systems*. To appear in the V Brazilian Symposium on Computing System Engineering (SBESC), 2015. **(Aceito)**