Universidade Federal do Amazonas

Faculdade de Tecnologia

Programa de Pós-Graduação em Engenharia Elétrica

Avaliação de Projetos de Filtros Digitais de Ponto-Fixo usando Teorias do Módulo da Satisfatibilidade

Renato Barbosa Abreu

Manaus – Amazonas

Junho de 2014

Renato Barbosa Abreu

Avaliação de Projetos de Filtros Digitais de Ponto-Fixo usando Teorias do Módulo da Satisfatibilidade

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica do Departamento de Eletrônica e Computação da Universidade Federal do Amazonas, como requisito para obtenção do Título de Mestre em Engenharia Elétrica. Área de concentração: Controle e Automação de Sistemas.

Orientador: Prof. Ph.D. Lucas Carvalho Cordeiro

Ficha Catalográfica
(Catalogação realizada pela Biblioteca Central da UFAM)

Abreu, Renato Barbosa

A162a Avaliação de projetos de filtros digitais de ponto-fixo usando teorias do módulo da satisfatibilidade / Renato Barbosa Abreu, 2014.

74f.

Dissertação (mestrado em Engenharia Elétrica) – Universidade Federal do Amazonas.

Orientador: Prof. Ph.D. Lucas Carvalho Cordeiro

1. Filtros digitais 2. Automação 3. Engenharia elétrica I. Cordeiro, Lucas Carvalho (Orient.) II. Universidade Federal do Amazonas III. Título

CDU(2007) 621.3:004(043.3)

Renato Barbosa Abreu

Avaliação de Projetos de Filtros Digitais de Ponto-Fixo usando Teorias do Módulo da Satisfatibilidade

Banca Examinadora

Prof. Ph.D. Lucas Carvalho Cordeiro – Presidente e Orientador
Departamento de Eletrônica e Computação – UFAM

Prof. Ph.D. Eduardo Antônio Barros da Silva
Departamento de Eletrônica – UFRJ/COPPE

Prof. D.Sc. Eddie Batista de Lima Filho
Centro de Ciência, Tecnologia e Inovação do Pólo Industrial de Manaus – CT-PIM

Manaus – Amazonas
Junho de 2014

À Deus, à família e aos amigos.

Agradecimentos

Esta pesquisa foi apoiada pelo concessão do CNPq 475647/2013-0. Além disso, os responsáveis por este trabalho gostariam de agradecer ao PPGEE/UFAM por apoiar o desenvolvimento do projeto. Esta pesquisa também foi apoiada pela concessão do CNPq e FAPEAM. O desenvolvimento do ESBMC é financiado pelo *British Council* e teve o apoio do *Instituto Nokia de Tecnologia* (INdT).

"Pensem o que quiserem de ti; faz aquilo que te parece justo"

Pitágoras (570-495 BC)

Resumo

Atualmente, os filtros digitais são empregados em uma ampla variedade de aplicações para processamento de sinais, utilizando tanto processadores de ponto flutuante quanto de ponto fixo. No que diz respeito a este último, algumas implementações de filtro podem estar mais propensas a erros, devido a problemas relacionados com a palavra de dados de comprimento finito. Em particular, o processamento de sinais utilizando tais realizações pode produzir o problema de estouro aritmético e ruídos indesejados causados pela quantização e efeitos de arredondamento, durante operações acumulativas de adição e multiplicação. O presente trabalho aborda este problema e propõe uma nova metodologia para a verificação de filtros digitais, com base em um verificador de modelos no estado da arte, chamado ESBMC, que suporta linguagens C/C++ e emprega solucionadores baseados em teoria do módulo da satisfatibilidade. Além de verificar a ocorrência de estouro aritmético e ciclo limite, a presente abordagem também pode verificar propriedades de projeto, como estabilidade e resposta em frequência, bem como restrições temporais e erro de saída, com base em modelos de tempo discreto implementados em C. Os experimentos realizados durante este trabalho mostram que a metodologia proposta é eficaz, pois encontra erros de projeto realistas, que estão relacionados a implementações de filtros digitais em ponto fixo. Vale ressaltar que os resultados apresentados evidenciam que o método proposto, além de auxiliar o projetista a determinar o número de bits da representação de ponto fixo, também pode ajudar a definir detalhes de realização e estrutura de filtro.

Palavras-chave: filtros em ponto-fixo, métodos formais, verificação de modelos

Abstract

Currently, digital filters are employed in a wide variety of signal processing applications, using floating- and fixed-point processors. Regarding the latter, some filter implementations may be prone to errors, due to problems related to finite word-length. In particular, signal processing modules present in such realizations can produce overflows and unwanted noise caused by the quantization and round-off effects, during accumulative-addition and multiplication operations. The present work addresses this problem and proposes a new methodology to verify digital filters, based on a state-of-the-art bounded model checker called ESBMC, which supports full C/C++ and employs satisfiability-modulo-theories solvers. In addition to verifying overflow and limit-cycle occurrences, the present approach can also check design properties, like stability and frequency response, as well as output errors and time constraints, based on discrete-time models implemented in C. The experiments conducted during this work show that the proposed methodology is effective, when finding realistic design errors related to fixed-point implementations of digital filters. It is worth noting that the proposed method, in addition to helping the designer to determine the number of bits for fixedpoint representations, can also aid to define details of filter realization and structure.

Keywords: fixed-point filters, formal methods, model checking

Abreviações

API - **A**pplication **P**rogramming **I**nterface

BMC - **B**ounded **M**odel **C**hecker

CPU - **C**entral **P**rocessing **U**nit

DSP - **D**igital **S**ignal **P**rocessor

ESBMC - **E**fficient **S**MT-Based **B**ounded **M**odel **C**hecker

FIR - **F**inite **I**mpulse **R**esponse

FPGA - **F**ield **P**rogrammable **G**ate **A**rray

IDE - **I**ntegrated **D**evelopment **E**nvironment

IIR - **I**nfinite **I**mpulse **R**esponse

RISC - **R**educed **I**nstruction **S**et **C**omputing

SAT - **Sat**isfiability

SMT - **S**atisfiability **M**odulo **T**heory

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

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

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Em processamento digital de sinais, um filtro digital é um sistema que executa operações em sinais discretos, de modo a modificar ou melhorar alguns dos seus aspectos, como nível de ruído e largura de banda. Um filtro pode ser classificado em dois tipos: de resposta ao impulso infinita (*infinite impulse response* - IIR) ou de resposta ao impulso finita (*finite impulse response* - FIR), comumente diferindo pela existência ou não de realimentação, respectivamente. Um dos procedimentos mais comuns realizados com filtros digitais, por exemplo, é reduzir ou alterar a largura de banda do sinal, com o objetivo de descartar informações indesejadas.

Filtros digitais têm sido amplamente utilizados em uma grande variedade de aplicações, devido principalmente à sua flexibilidade, aliada a uma baixa complexidade computacional, o que é reforçado pela disponibilidade de processadores digitais de sinais (*digital signal processor* - DSP) e arranjos de portas programável em campo (*field programmable gate arrays* - FPGAs). Esses dispositivos podem ser classificados em duas categorias: de ponto fixo ou flutuante, que se referem ao formato usado para armazenar e processar as representações de dados numéricos. Na aritmética de ponto fixo, os intervalos entre os números adjacentes representados são normalmente iguais; a aritmética de ponto flutuante, por sua vez, resulta em intervalos não uniformes, que podem ser até dez milhões de vezes menores que a magnitude do número, para um mesmo comprimento de palavra [1].

Recentemente, a disponibilidade de processadores de ponto flutuante aumentou substancialmente. Todavia, a alta velocidade dos processadores de ponto fixo, em combinação com o seu custo reduzido, ainda os tornam a escolha preferencial para projetos

de filtros digitais embarcados. No entanto, efeitos de quantização não-lineares, erros de arredondamento e estouros aritméticos (*overflow*), se manifestam mais gravemente em implementações de ponto fixo. Todos esses efeitos são causados por operações consecutivas de adição e multiplicação, utilizando palavras de comprimento finito (sequências de bits de tamanho fixo), o que pode afetar o comportamento do filtro desejado. Por exemplo, em relação a estruturas na forma direta, uma pequena mudança nos coeficientes do filtro, devido à quantização, pode resultar numa grande alteração na localização dos pólos e zeros do sistema [2].

Além disso, diferentes tipos de filtro apresentam diferentes problemas. Por exemplo, os filtros IIR podem sofrer de graves oscilações na saída, mesmo para um sinal de entrada nulo, o que é um conhecido como *ciclo limite* [3]. Filtros FIR, por sua vez, não manifestam tais efeitos do ciclo limite, contudo também sofrem outros problemas causados por limitações da palavra de comprimento finito (por exemplo, modificação de resposta de frequência). Existem muitos estudos sobre os efeitos de quantização e ciclo limite em filtros digitais, juntamente com técnicas para reduzir os seus efeitos, como previamente relatado por Claasen et al. [4]. No entanto, essas técnicas, como mudança de escala e saturação, normalmente resultam em algumas desvantagens, tais como o aumento da potência de ruído causado por erros de quantização e arredondamentos. Sendo assim, a magnitude do erro deve ser verificada, a fim de assegurar que está em um nível aceitável.

Outra propriedade importante, que surge durante a implementação de filtros digitais para aplicações em tempo real, é a restrição temporal [5]. Como amplamente conhecidos, os microcontroladores modernos e DSPs podem ser programados em linguagens de alto nível, como C e C++. Sendo assim, o código do filtro é compilado para instruções de baixo nível, que consomem ciclos de *clock* e devem atender a algumas restrições, de acordo com a frequência de amostragem do sistema e também do *buffer* disponível.

Normalmente, os projetistas de filtro empregam ferramentas avançadas para definir os parâmetros do sistema, de acordo com a operação desejada no domínio do tempo ou da frequência, e usam software de simulação para validar seu comportamento, juntamente com testes extensivos. No entanto, na maioria dos casos, a aritmética de números reais é considerada durante os cálculos, o que pode levar a suposições erradas sobre o desempenho do filtro implementado.

Existem algumas ferramentas para simular sistemas usando aritmética de ponto fixo [6, 7], que podem ser usadas durante a fase de projeto do filtro. Como exemplo, Sung e Kum [8] propõem algoritmos de busca para determinar o comprimento mínimo da palavra binária, através de uma abordagem baseada em simulações. No entanto, tais simulações (e testes) podem considerar um número limitado de cenários e entradas, que normalmente não exploram todos os comportamentos possíveis que um sistema pode apresentar. Assim, somente análises gráficas no domínio da frequência e simulações podem não ser suficientes para concluir sobre possíveis problemas relacionados à palavra de comprimento finito, bem como limitações de tempo de processamento do filtro.

Recentemente, Cox et al. [9] propuseram um método para a verificação de implementações de filtros IIR em ponto fixo, baseado em verificação de modelos limitada (*Bounded Model Checking* - BMC) utilizando solucionadores de teoria do módulo da satisfatibilidade (*Satisfiability Modulo Theory* - SMT), com o objetivo de checar as condições de verificação. A principal ideia da verificação de modelos baseada em SMT é considerar contra-exemplos de um tamanho específico k e gerar uma fórmula em lógica de primeira ordem, que pode ser satisfeita se e somente se esse contra-exemplo existir [10].

1.1 Descrição do Problema

No desenvolvimento de um sistema em ponto fixo, o projetista precisa resolver o dilema da definição da quantidade de bits necessária para representação numérica. Com relação a esse problema, deve-se levar em consideração os requisitos relacionados às seguintes métricas:

- Precisão: Quanto maior for a quantidade de bits para a representação numérica, menor será o erro de quantização;
- Desempenho: Uma maior quantidade de bits para representação numérica eleva o tempo de execução das operações aritméticas;
- Custo: Plataformas que suportam um maior número de bits, ou mais memória, possuem custo mais elevado.

Sendo assim o projetista deve procurar um balanço entre esses requisitos e determinar o comprimento mínimo da palavra para representação da parte inteira e da parte fracionária dos valores. Através de métodos analíticos, estima-se o valor mínimo e máximo do intervalo que deve ser representável no sistema. Porém, dependendo da complexidade do filtro, pode ser complicado determinar esses valores através de cálculos, sendo necessário fazer aproximações por métodos numéricos. Outras abordagens utilizadas são simulações e testes extensivos, nos quais o filtro é submetido a determinadas entradas e, a partir dos resultados máximos e mínimos obtidos, define-se o comprimento da palavra binária. No entanto, esses testes podem não explorar todas as possíveis entradas existentes, o que leva a alguns estados de falha. Uma alternativa para se evitar possíveis falhas seria aumentar deliberadamente o comprimento da palavra para representação numérica. Entretanto, isso pode elevar o custo do sistema, ou tornar impraticável a aplicação da solução em um sistema em tempo real (*hard real-time*), no qual é imprescindível que a resposta ocorra dentro do tempo determinado [11], devido a maior duração na execução das operações.

1.2 Objetivos

O objetivo geral deste trabalho é apresentar uma forma de detectar problemas no projeto de filtros digitais, implementados em ponto fixo e utilizando um verificador de modelos limitado baseado em teoria da satisfatibilidade (abreviado em inglês, *SMT-based BMC*). Esse tipo de verificação pode evitar que falhas graves, como *overflow*, instabilidade e atraso de resposta aconteçam durante a operação de tais sistemas em campo.

Os objetivos específicos são listados a seguir:

- Pesquisar um método para verificação de *overflow*, ciclo limite, restrições temporais e erro de saída de filtros digitais, em ponto fixo, através de um software BMC baseado em SMT comercial;
- Incorporar verificações de resposta em frequência e de estabilidade, para execução através do software BMC;

- Propor um sistema capaz de verificar filtros implementados em diversas estruturas (Forma Direta I, Forma Direta II, Forma Direta Transposta II, em Cascata e em Paralelo) e com diferentes tamanhos de ponto fixo (número de bits para parte inteira e fracionária);
- Aprimorar a implementação para reduzir o tempo de verificação das propriedades, tornando a aplicação do método mais escalonável;
- Aplicar a metodologia proposta à verificação de *benchmarks* e, através dos resultados obtidos, concluir sobre a sua eficácia.

1.3 Contribuições

O presente trabalho aborda o problema mencionado anteriormente, na Seção 1.1, e descreve o uso de um verificador de modelos limitado baseado em SMT para programas em C/C++, a fim de verificar possíveis problemas causados pela aritmética de ponto fixo, em filtros digitais. Em relação ao método proposto, além de detectar problemas de *overflow* e ciclo limite, as seguintes contribuições podem também ser encontradas:

- O tempo de processamento associado é considerado durante a execução da função do filtro, o qual verifica o tempo máximo aceitável das operações;
- A especificação da resposta em frequência pode ser verificada, considerando-se os efeitos da quantização nos coeficientes de filtro;
- A localização dos pólos e zeros pode ser verificada, a fim de concluir sobre a estabilidade do sistema, dada a quantização dos parâmetros;
- A magnitude do erro de saída é verificada, com base em um modelo de maior precisão e utilizando-se lógica de primeira ordem, de modo a determinar se o erro do sistema está dentro de uma margem aceitável;
- O BMC é explorado para verificar o código do filtro digital em C, o qual se destina a ser incorporado em micro-controladores e DSPs.

Vale a pena ressaltar que este último é mais próximo de implementações reais, onde construções específicas da linguagem C (por exemplo, a aritmética de ponteiros e comparações) são usadas no desenvolvimento de filtros digitais. Além disso, o uso de BMC baseado em SMT, associado ao projeto de filtros digitais, não é muito difundido entre os desenvolvedores da área de processamento de sinais, o que significa que este trabalho pode potencialmente gerar valor para eles.

1.4 Organização da Dissertação

Neste capítulo, são descritos o contexto, a motivação e os objetivos deste trabalho. Os próximos capítulos deste texto estão organizados da seguinte forma:

Inicialmente, o Capítulo 2 discute os trabalhos relacionados, reunindo algumas referências sobre o tema. Alguns trabalhos sobre verificação de modelos, verificação temporal, teste e verificação de filtros digitais são destacados.

O Capítulo 3 faz uma breve introdução sobre a realização de filtros digitais, enfatizando alguns aspectos relacionados à implementação em processadores em ponto fixo, relacionados a estruturas e formas de representação. Alguns fundamentos sobre lógica e verificação limitada de modelos baseada em SMT também são abordados, pois foram utilizados neste trabalho, para a verificação de filtros digitais.

O Capítulo 4 descreve a metodologia e os algoritmos implementados para a verificação de *overflow*, ciclo limite, resposta em frequência, estabilidade, restrição temporal e erro de saída de filtros.

No Capítulo 5, apresentam-se as configurações utilizadas para executar os testes e os resultados dos experimentos executados, para diferentes tipos de filtros digitais, através de modelos implementados em linguagem C.

Por fim, o Capítulo 6 apresenta as conclusões, destacando a importância da verificação BMC de filtros em ponto fixo.

Capítulo 2

Trabalhos Relacionados

Este capítulo menciona alguns trabalhos que possuem objetivos semelhantes aos apresentados pelo método proposto, com o objetivo de dar uma noção sobre o estado das técnicas atualmente utilizadas. O capítulo está dividido em duas seções, sendo que a Seção 2.1 apresenta trabalhos relacionados à verificação de programas e sistemas embarcados, enquanto que a Seção 2.2 menciona os trabalhos que tratam sobre a verificação de problemas em filtros digitais.

2.1 Verificação de Programas

A aplicação de ferramentas BMC baseadas em SMT está se tornando popular para a verificação de software, principalmente devido ao advento de solucionadores SMT sofisticados, que foram construídos sobre solucionadores de satisfatibilidade booleana eficientes [12], [13], [14]. Trabalhos anteriores, relacionados a BMCs baseados em SMT, abordam o problema de verificar programas em C que usam operações de bit, aritmética de ponto fixo, aritmética de ponteiro e comparações [10], [15]. No entanto, existem poucos trabalhos que abordam o problema da verificação de modelos relacionados a implementações de filtros digitais, no tocante a propriedades de segurança e vivacidade. No que concerne às propriedades de segurança, deve se garantir que nenhuma situação indesejada aconteça (por exemplo, um *overflow*). Já quanto às propriedades de vivacidade, o que se espera é que uma situação desejada aconteça eventualmente (por exemplo, a saída se estabilize dada uma entrada constante). Definições mais detalhadas sobre propriedades

de segurança e vivacidade são descritas por Baier et al. [16].

Para a verificação eficiente de programas utilizados em sistemas embarcados, Cordeiro et al. [10] propõem o ESBMC, um verificador de modelos aperfeiçoado que fornece suporte mais preciso para as variáveis de largura de bits finita, operações com vetores de bits, vetores, estruturas, uniões e ponteiros. No entanto, os autores não consideram a verificação de implementações com palavras de comprimentos altamente otimizados (isto é, a menor quantidade de bits para a representação das partes inteira e fracionária, de números em ponto fixo), suportando apenas as larguras de 16, 32 e 64 bits. Neste trabalho, a aplicação do verificador de modelos ESBMC foi estendida para verificar filtros otimizados, com redução do número de bits para a sua representação em ponto fixo. Além disso, verificações de *overflow* para números de ponto fixo, de diferentes tamanhos, foram abordadas.

Para sistemas de tempo real (ou *hard real time*), a análise do pior caso do tempo de execução (*Worst-case execution time* - WCET) é normalmente aplicada para garantir que os requisitos de tempo sejam cumpridos. Kim et al. [17] apresentam uma abordagem híbrida para estender a análise de WCET, utilizando um verificador de modelos. Barreto et al. [18] utilizam o ESBMC para verificar restrições de tempo em softwares C embarcados, no nível de funções. O presente trabalho, no entanto, vai mais a fundo na verificação, a fim de estabelecer os períodos no nível de instruções, o que possibilita definir melhor as assertivas das restrições, que por sua vez estão associados ao tempo de processamento do filtro e a taxa de amostragem especificada.

2.2 Verificação de Filtros Digitais

Desde o início da utilização de sistemas digitais, os problemas de implementação de filtros, com aritmética de ponto fixo, já eram um tema de estudos. Modelos estatísticos foram propostos, com o objetivo de analisar ruído e possibilitar uma comparação com sistemas em ponto flutuante, como descrito por Weinstein e Oppenheim [19]. Os mesmos autores também destacam os efeitos da implementação de equações de diferenças com registradores de comprimento finito [20], como erros causados por arredondamentos, quantização dos coeficientes e *overflow*. Os modelos estatísticos para o ruído de

arredondamento, formulados para filtros digitais simples, podem ser aplicados em sistemas mais complexos, que os utilizam como blocos construtivos.

Bauer et al. [21][22] mostraram a utilização de métodos computacionais baseados em busca exaustiva, com o objetivo de determinar a ausência de ciclos limite, quando da utilização de filtros digitais na forma direta. Logo, técnicas baseadas em simulação e testes extensivos se tornaram bastante utilizadas, principalmente na verificação de filtros digitais, como apresentado por Bailey [23] e também por Raheem et al. [24]. A principal ideia nesses trabalhos é auxiliar a análise da representação numérica, além de estruturas utilizadas nas implementações de filtros, a fim de reduzir o ruído de arredondamento e atender às especificações de projeto.

Já Akbarpour et al. [25] propõem uma metodologia para a análise de erros causados pelo tamanho limitado da palavra, em filtros digitais, através de um provador de teoremas de lógica de ordem superior (*Higher Order Logic* - HOL). Nesse trabalho foram utilizadas teorias de HOL para modelar a especificação do filtro ideal real, seguido pelas implementações correspondentes baseadas no padrão IEEE de ponto flutuante e, também, em aritmética de ponto fixo utilizando lógica de ordem superior. Então, aplicam-se funções para avaliar a diferença entre a saída real e as saídas das implementações em ponto flutuante e em ponto fixo, além de analisar os efeitos do erro de arrendondamento acumulado, em filtros digitais. Por fim, o autor destaca a existência de trabalhos que datam desde a década de sessenta e utilizam demonstrações teóricas e simulações, mas que aquele seria o primeiro trabalho a utilizar verificação formal baseada em HOL, para a avaliação de filtros digitais.

Cox et al. [9] introduziram uma nova abordagem, que utiliza a análise por precisão de bits para verificação de implementações de filtros digitais em ponto fixo, com base na verificação de modelos limitada e solucionadores SMT. Eles mostram que a abordagem por precisão de bits é mais eficiente e produz menos alarmes falsos, quando comparada a solucionadores com aritmética real. Em seguida, os mesmos autores estenderam a técnica anterior, considerando desta vez um número ilimitado de iterações no sistema, podendo assim efetuar a prova da corretude de implementações de filtros digitais [26]. O sistema proposto foi implementado em linguagem *OCaml* [27], com chamadas diretas à interface de programação do Z3 [28], que é um provador de teoremas baseado em SMT.

Isso cria algumas dificuldades em relação a extensão e customização do método, já que o desenvolvimento utilizando essas ferramentas não é muito conhecido, quando comparado ao desenvolvimento em uma linguagem como ANSI-C. Então, caso os modelos de filtros definidos no sistema proposto não estejam próximos ao cenário de uma determinada implementação real, a adaptação da ferramenta torna-se difícil.

Alguns trabalhos sobre a verificação de filtros têm se concentrado em encontrar o comprimento adequado da palavra binária, para a implementação em ponto fixo. Fang et al. [29] apresentam uma nova técnica que se baseia em métodos de representação de intervalos da aritmética afim, cuja abordagem é capaz de analisar projetos maiores e sistemas realimentados de uma forma mais sistemática e precisa. No entanto, tal como indicado por Cox et al. [9], a utilização de técnicas conservadoras baseadas em aritmética afim podem produzir alarmes falsos. Algoritmos de busca, com o objetivo de determinar o limite mínimo do comprimento da palavra, também são apresentados por Sung et al. [8]. No entanto, os autores adotam uma abordagem baseada em simulação e, portanto, não exploram todas os possíveis comportamentos que o filtro digital pode apresentar.

A Tabela 2.1 mostra um comparativo entre os principais trabalhos relacionados a análise e verificação de filtros digitais. No presente trabalho, utilizou-se a mesma abordagem de verificação limitada proposta por Cox [9], porém, os filtros digitais foram implementados na linguagem C e uma ferramenta BMC comercial foi utilizada. Além disso, a técnica foi estendida para verificar limitações de tempo em aplicações embarcadas implementadas em C, o que é, portanto, mais próximo do cenário real. Vale ressaltar também que propriedades de projeto como resposta em frequência, estabilidade e erro de saída foram consideradas, o que contou ainda com uma análise quanto à forma de realização do filtro.

Tabela 2.1: Comparativo resumido entre principais trabalhos relacionados.

Trabalhos relacionados	**Simulações e métodos estatísticos**	**Método formal**	**Verificação ilimitada**	**Verificação de erro de saída**	**Verificação de overflow e ciclo limite**	**Verificação de resp. frequência, estabilidade e temporização**
[19], [20]	X			X		
[21]	X				X	
[23], [24], [29]	X			X		X
[25]		X		X		
[9]		X			X	
[26]		X	X		X	
Trabalho proposto		X		X	X	X

Capítulo 3

Fundamentação Teórica

Neste capítulo, alguns conceitos sobre filtros digitais serão introduzidos, o que compreende as suas estruturas e os efeitos devido à utilização de uma quantidade limitada de bits para a representação numérica. O formato e as restrições inerentes à representação em ponto-fixo também serão abordadas. Além disso, alguns conceitos sobre lógica proposicional e teoria da satisfatibilidade serão apresentados, devido ao fato de serem utilizados em solucionadores SMT, que compõem a base da ferramenta de verificação adotada.

3.1 Fundamentos sobre Filtros Digitais

Um filtro digital pode ser definido como um sistema linear discreto e invariante no tempo, geralmente descrito por uma equação de diferenças com coeficientes constantes, como

$$y(n) = -\sum_{k=1}^{N} a_k y(n-k) + \sum_{k=0}^{M} b_k x(n-k), \tag{3.1}$$

onde $y(n)$ é a saída no instante n, $y(n-k)$ é a saída a k instantes anteriores, $x(n-k)$ são as entradas k instantes anteriores, a_k são os coeficientes para as saídas anteriores, b_k são os coeficientes para as entradas anteriores, N é o número de saídas anteriores e M é o número de entradas do sistema. O projeto de um filtro digital consiste em encontrar os valores dos coeficientes a_k e b_k, que produzem a resposta em frequência esperada.

O sistema descrito pela Equação (3.1) pode também ser representado através de uma função racional, largamente conhecida como *função de transferência* do sistema. Essa função pode ser deduzida através da transformada Z aos dois lados da Equação (3.1),

juntamente com a aplicação do teorema de deslocamento no tempo [30]. A partir daí, é possível chegar à representação que relaciona a saída $Y(z)$ à entrada $X(z)$, como descrito pela Equação (3.2)

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}}. \tag{3.2}$$

Tal equação descreve de forma geral os filtros recursivos (IIR). Para filtros não recursivos (FIR), basta simplificar a equação, fazendo-se $a_k = 0$ para $1 \leq k \leq N$, o que resulta em

$$H(z) = \sum_{k=0}^{M} b_k z^{-k}. \tag{3.3}$$

Considerando-se o sistema expresso como uma razão de polinômios, as raízes do numerador são referidas como os *zeros* do sistema, ao passo que as do denominador são referidas como os *pólos* do sistema. Então, a função de transferência do sistema pode ser expressa através de seus pólos p_k e zeros z_k, como

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}} = \frac{b_0}{a_0} z^{N-M} \frac{\prod_{k=0}^{M} (z - z_k)}{\prod_{k=1}^{N} (z - p_k)}. \tag{3.4}$$

A função $H(z)$ pode ser representada graficamente em diagrama de pólos e zeros, no plano complexo, como ilustrado na Figura 3.1. A região marcada preenche o círculo de raio unitário. Para que um sistema causal (cuja saída em um dado instante não depende de entradas em instantes posteriores [31]) seja estável, todos os pólos do sistema devem ter módulo menor do que 1, ou seja, devem estar localizados dentro do círculo de raio unitário.

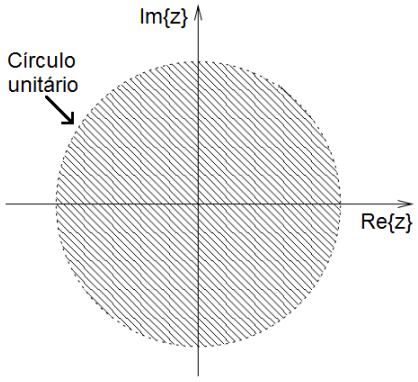
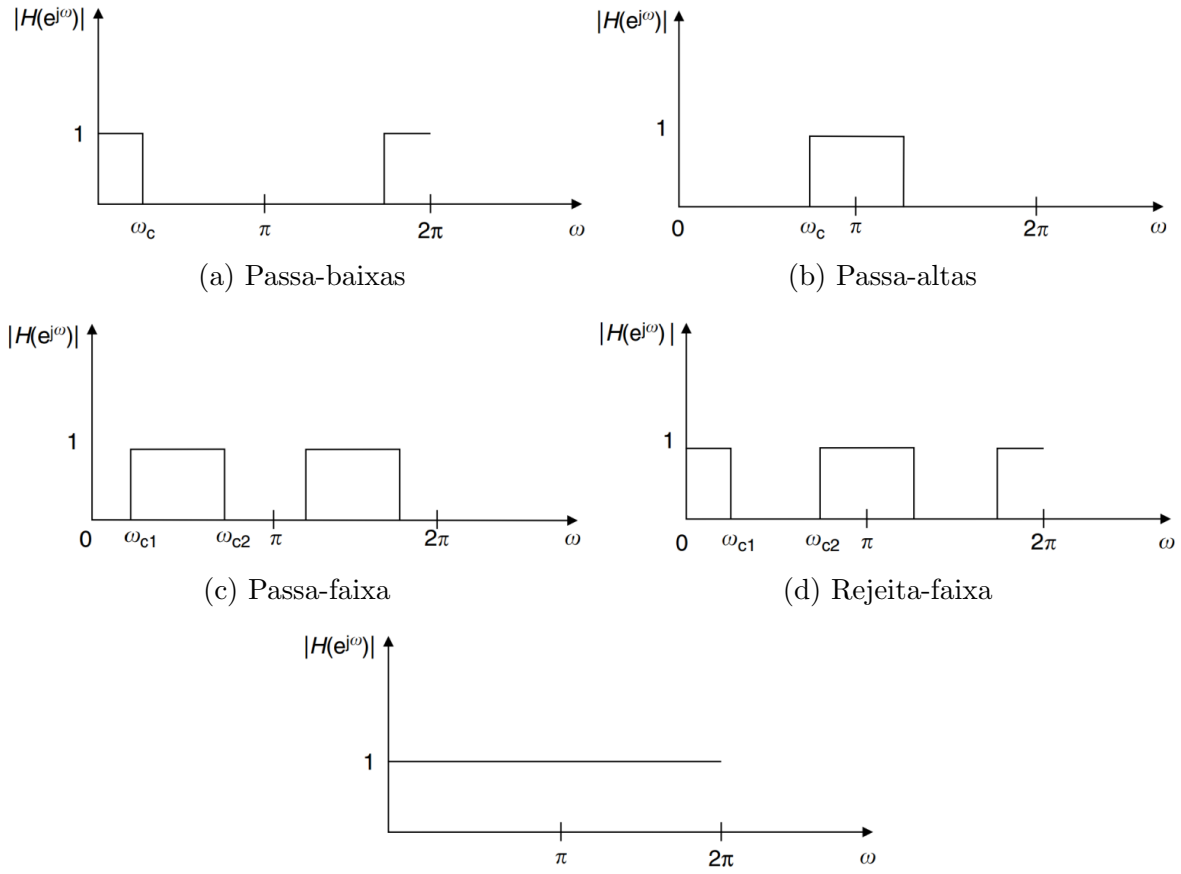


Figura 3.1: Plano complexo do diagrama de pólos e zeros.

Normalmente, os filtros são classificados de acordo com suas características ideais, no domínio da frequência, tais como

- Passa-baixas - atenua frequências maiores que a frequência de corte;
- Passa-altas - atenua frequências menores que a frequência de corte;
- Passa-faixa - atenua frequências fora do intervalo de frequências de corte;
- Rejeita-faixa - atenua frequências entre o intervalo de frequências de corte;
- Passa-tudo - não atenua frequências, mas modifica a fase do sinal.

As características da resposta em magnitude ideais desses tipos de filtros estão ilustradas na Figura 3.2.



(a) Passa-baixas

(b) Passa-altas

(c) Passa-faixa

(d) Rejeita-faixa

(e) Passa-tudo

Figura 3.2: Resposta em magnitude de alguns filtros ideais.

Além dos valores de ganhos e frequências de corte, o projeto de filtros FIR e IIR pode envolver outras especificações, no domínio da frequência, tais como tamanho da banda de transição, amplitude da ondulação na banda passante e na banda de corte. Contudo, não

faz parte do escopo deste trabalho apresentar métodos de projeto de filtros IIR e FIR. Este é um tema extenso, coberto em livros de processamento digital de sinais [2], [30], [31].

3.2 Implementação de Filtros em Ponto-Fixo

Existem muitas maneiras de implementar a Equação (3.1) em um dispositivo digital programável, seja esta em hardware ou software, dependendo da estrutura do sistema. A Forma Direta I de realização de sistemas IIR está ilustrada na Figura 3.3, cuja formulação é bastante simples. Essa estrutura é obtida conectando o sistema direto formado pelos coeficientes b_k, em cascata com o sistema realimentado formado pelos coeficientes a_k. Esta realização do filtro, no entanto, requer um número maior de posições de memória que a Forma Direta II [2], mostrada na Figura 3.4. Entretanto, uma vez que este número é bem conhecido pelos projetistas $(M + N)$, o problema de verificação de memória não é abordado no presente trabalho.

As Figuras 3.3, 3.4 e 3.5 exibem as formas comuns de estruturas de filtros digitais. É possível notar que os atrasos são inseridos pelos ramos marcados com o bloco z^{-1}.

As estruturas das Figuras 3.3 e 3.4 são chamadas de "forma direta", pois são obtidas diretamente da função de transferência $H(z)$, sem nenhum rearranjo. Já a estrutura da Figura 3.5, por sua vez, é obtida através a transposição da estrutura na forma direta II [2].

Além das formas diretas, outras estruturas podem ser obtidas através de agrupamen-

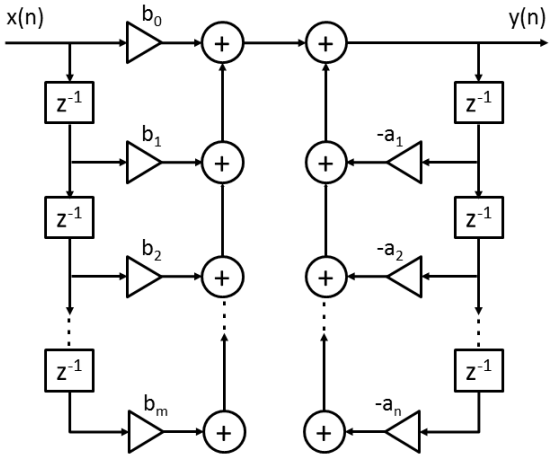


Figura 3.3: Estrutura de um Filtro IIR na Forma Direta I.

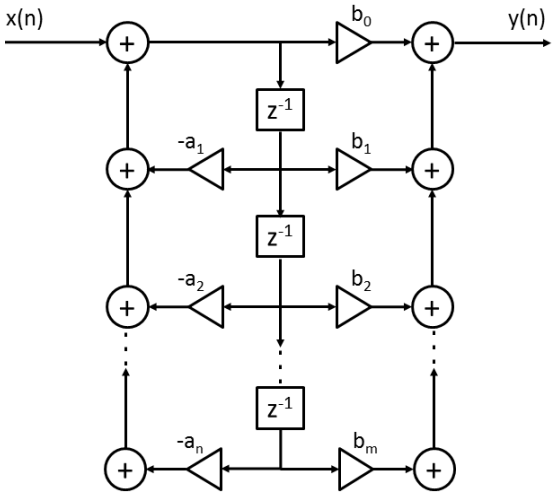


Figura 3.4: Estrutura de um Filtro IIR na Forma Direta II.

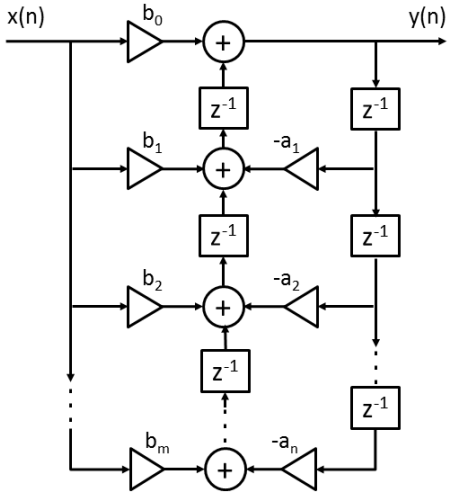


Figura 3.5: Estrutura de um Filtro IIR na Forma Direta Transposta II.

tos em Cascata, como mostrado na Figura 3.6a, e em Paralelo, como na Figura 3.6b. Os subsistemas $H_k(z)$ podem ser obtidos através de fatoração ou expansão em frações parciais, de uma função de transferência de um sistema de maior ordem.

Neste trabalho, para a demonstração do método proposto, os modelos das estruturas na Forma Direta I, Forma Direta II e Forma Direta Transposta II foram desenvolvidos em linguagem C. Também foram verificados modelos de estruturas em cascata e em paralelo, que foram construídos utilizando blocos de segunda ordem, nas formas diretas.

Na realização de filtros digitais em ponto fixo, os coeficientes e os resultados dos cálculos intermediários sofrem o efeito da quantização e erros de arredondamento.

Definição 3.1. *Um quantizador é um elemento, dispositivo ou função que faz com que*

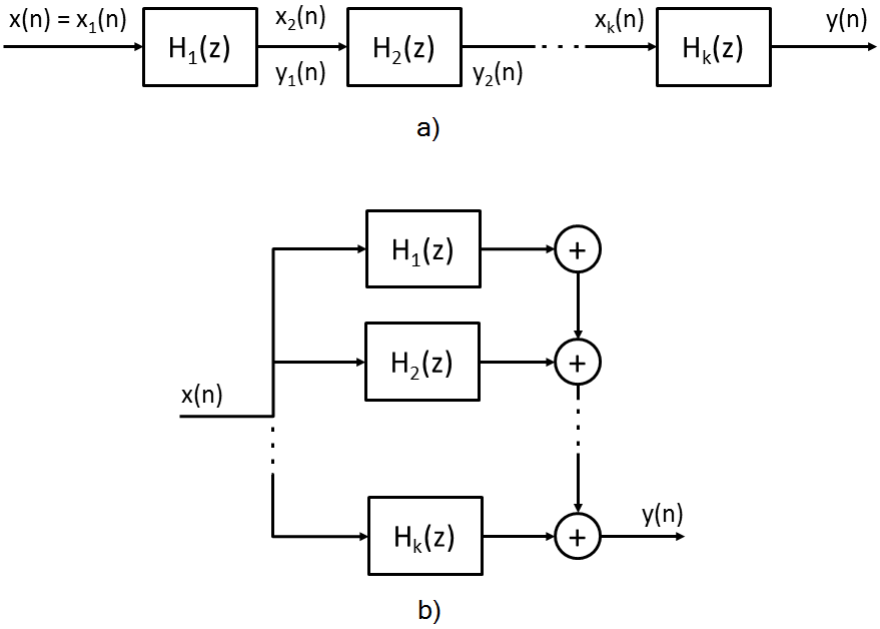


Figura 3.6: a)Estrutura de Filtro (a) em Cascata e (b) em Paralelo.

valores de um sinal sejam aproximados por valores de um conjunto discreto finito.

Aqui, o quantizador de arredondamento Q(x) é considerado. Para este quantizador, o erro máximo causado pelo arredondamento é de 2^{-l-1}, onde l é o número de bits da parte fracionária.

No caso em que o resultado de uma adição ou multiplicação excede a quantidade de bits disponíveis para a representação numérica, há um *overflow*.

Definição 3.2. *Um overflow acontece quando um valor excede os limites do intervalo de representação numérica, em um sistema digital.*

Para a verificação do ciclo limite, é permitido que o efeito de *overflow* aconteça sem interromper a verificação. Sendo assim, caso ocorra um *overflow*, o resultado é contornado dentro do intervalo representável (*wrap-around*). Considera-se a aritmética do complemento de dois, como representação binária dos números.

Definição 3.3. *O wrap-around ocorre quando um resultado fora do intervalo de representação numérica passa do valor máximo para o mínimo, ou a partir de um mínimo até ao máximo, após um overflow.*

A Figura 3.7 mostra o comportamento do quantizador de arredondamento e o efeito do *wrap-around*, quando um valor chega no limite de sua representação em ponto-fixo.

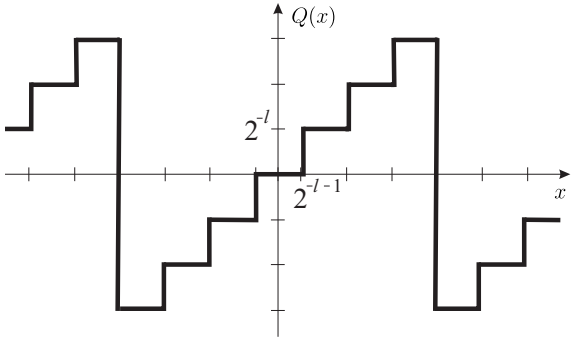


Figura 3.7: Saída do quantizador de arredondamento de l bits, com *wrap-around*

Para se obter um modelo mais realista do sistema de precisão finita, é necessário considerar a quantização de cada valor numérico no sistema, o que inclui as entradas, os coeficientes e os resultados de operações aritméticas. A Figura 3.8 mostra esse modelo, para um filtro de um pólo.

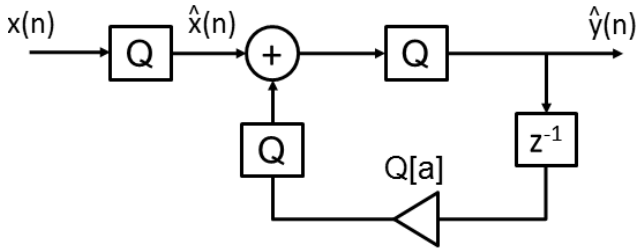


Figura 3.8: Modelo realista de um filtro de único pólo, com quantização.

Geralmente, várias abordagens de aproximações são adotadas para simplificar a análise de sistemas não lineares. Por exemplo, a caracterização estatística dos efeitos de quantização é usada na determinação do comprimento de palavra mínimo, de modo que se atinja o desempenho desejado para o sistema [2]. Neste trabalho, no entanto, uma outra abordagem baseada em métodos formais é apresentada. Através de uma ferramenta BMC, todas as entradas possíveis são aplicadas, com o objetivo de identificar os potenciais problemas do sistema.

3.3 Representação em Ponto-Fixo

Para representar números em formato de ponto fixo, utiliza-se um par de dígitos separados por um ponto decimal. Os dígitos à esquerda representam a parte inteira, ao passo que os dígitos à direita representam a parte fracionária do numero. O sistema de complemento de dois é o método mais comum para representar números com sinal, em processadores de ponto fixo. Nesse sistema, o número real X, descrito pelo posicionamento de ponto fixo $\langle k; l \rangle$ do número $(b_{k-1}\ b_{k-2}\ ...\ b_1\ b_0\ \cdot\ b_{-1}\ b_{-2}\ ...\ b_{-l})$, pode ser representado de acordo com

$$X = -b_{k-1}2^{k-1} + \sum_{i=k-2}^{-l} b_i 2^i. \tag{3.5}$$

O bit mais significativo $-b_{k-1}$ é usado para o sinal. Assim, o valor máximo representável por um número, que é constituído por uma parte inteira com k bits e uma parte fracionária com l bits, é $2^{k-1} - 2^{-l}$, com valor mínimo -2^{k-1}. O quantizador, representado na Figura 3.8 pelo bloco Q, arredonda os números dentro dessa faixa. Se um número não cabe no intervalo, então isso indica um *overflow*. Durante o processo de verificação, uma assertiva (*assert*) pode então detectar o *overflow* como uma falha no sistema, ou o quantizador pode contornar o resultado dentro do intervalo, como mostrado na Figura 3.7.

3.4 Fundamentos de Lógica Computacional

Esta seção introduz uma definição sobre lógica proposicional, incluindo sintaxe e semântica. Mais informações podem ser encontradas em livros texto, tais como o escrito por Bradley et al. [32] e também por Huth et al. [33]. A lógica pode ser definida por meio de símbolos e de um sistema de regras para manipular os símbolos. O uso da lógica permite modelar programas e avaliá-los formalmente [34]. A lógica proposicional é definida por uma relação binária, na qual se presume que toda sentença deve ser verdadeira (*tt* ou *true*) ou falsa (*ff* ou *false*).

Definição 3.4. *A sintaxe de uma fórmula em lógica proposicional é definida pela seguinte gramática:*

$$\begin{aligned} Fml &::= Fml \wedge Fml \mid \neg Fml \mid (Fml) \mid Atom \\ Atom &::= Variable \mid true \mid false \end{aligned}$$

Usando os operadores de conjunção ($\wedge$) e negação ($\neg$), é possível expressar todas as proposições lógicas. Outros operadores lógicos como disjunção ($\vee$), implicação ($\Rightarrow$), equivalência ($\Leftrightarrow$), ou-exclusivo ($\oplus$), e expressão condicional (*ite*, de *if-then-else*) podem ser definidos como segue.

Definição 3.5. *Os operadores lógicos usuais são definidos da seguinte forma:*

- $\phi_1 \vee \phi_2 \equiv \neg(\neg\phi_1 \wedge \neg\phi_2)$
- $\phi_1 \Rightarrow \phi_2 \equiv \neg\phi_1 \vee \phi_2$
- $\phi_1 \Leftrightarrow \phi_2 \equiv (\phi_1 \Rightarrow \phi_2) \wedge (\phi_2 \Rightarrow \phi_1)$
- $\phi_1 \oplus \phi_2 \equiv (\phi_1 \wedge \neg\phi_2) \vee (\phi_2 \wedge \neg\phi_1)$
- $ite(\theta, \phi_1, \phi_2) \equiv (\theta \wedge \phi_1) \vee (\neg\theta \wedge \phi_2)$

Uma fórmula em lógica proposicional ou proposição lógica é definida em termos dos elementos básicos *true*, *false*, uma variável proposicional x ou a aplicação de um dos seguintes operadores lógicos a uma fórmula ϕ: "não" ($\neg\phi$), "e" ($\phi_1 \wedge \phi_2$), "ou" ($\phi_1 \vee \phi_2$), "implica" ($\phi_1 \Rightarrow \phi_2$), "sse" ($\phi_1 \Leftrightarrow \phi_2$). "paridade" ($\phi_1 \oplus \phi_2$) or "ite" ($ite(\theta, \phi_1, \phi_2)$).

Definição 3.6. *Uma proposição lógica é uma fórmula bem formulada se forem usadas as regras de construção da Definição 3.4 para obtê-la, dado que a negação tem prioridade sobre a conjunção.*

Definição 3.7. *A precedência relativa dos operadores lógicos, do maior para o menor, é definida da seguinte forma:* $\neg$, $\wedge$, $\vee$, $\Rightarrow$ *e* $\Leftrightarrow$.

A fim de se verificar se uma dada proposição lógica é verdadeira ou falsa, primeiro define-se um mecanismo para avaliar as variáveis proposicionais, por meio de *interpretações*. Uma interpretação I atribui, a toda variável proposicional, exatamente um valor verdade. Por exemplo, $I = \{x_1 \mapsto tt, x_2 \mapsto ff\}$ é uma interpretação, atribuindo

verdadeiro a x_1 e *falso* a x_2. Dada uma proposição lógica e uma interpretação, o valor verdade de uma fórmula pode ser computado por uma tabela verdade ou por indução.

Uma definição indutiva da semântica da lógica proposicional também é descrita, que define o significado dos operadores básicos e também o significado de formulações mais complexas, em termos dos operadores básicos. Escreve-se $I \models \phi$ se ϕ resultar em tt sob I, e $I \not\models \phi$ se ϕ resultar em ff sob I.

Definição 3.8. *A resolução da fórmula ϕ, sob uma interpretação I, é definida a seguir:*

- $I \models x$ *sse* $I[x] = tt$
- $I \models \neg\phi$ *sse* $I \not\models \phi$
- $I \models \phi_1 \wedge \phi_2$ *sse* $I \models \phi_1$ *and* $I \models \phi_2$

Lema 3.1. *As semânticas de fórmulas mais complexas são avaliadas como:*

- $I \models \phi_1 \vee \phi_2$ *sse* $I \models \phi_1$ *ou* $I \models \phi_2$
- $I \models \phi_1 \Rightarrow \phi_2$ *sse, sempreque* $I \models \phi_1$ *então* $I \models \phi_2$
- $I \models \phi_1 \Leftrightarrow \phi_2$ *sse* $I \models \phi_1$ *e* $I \models \phi_2$, *ou* $I \not\models \phi_1$ *e* $I \not\models \phi_2$

Um algoritmo pode ser facilmente implementado, para decidir sobre a *satisfatibilidade* de uma proposição lógica.

Definição 3.9. *Uma fórmula em lógica proposicional é satisfazível em relação a uma classe de interpretações, se existir uma atribuição a suas variáveis na qual a fórmula resulte em verdadeiro.*

A entrada do algoritmo para verificar a satisfatibilidade é normalmente uma proposição lógica, na forma normal conjuntiva.

Definição 3.10. *Formalmente, uma proposição lógica ϕ está na forma normal conjuntiva se esta consistir de uma conjunção de uma ou mais cláusulas, em que cada cláusula é uma disjunção de um ou mais literais. Sua forma é $\bigwedge_i \left(\bigvee_j l_{ij} \right)$, onde cada l_{ij} é um literal.*

O problema da satisfatibilidade proposicional (SAT) é então decidir se existe uma atribuição satisfazível para os literais da proposição lógica ϕ, na forma normal conjuntiva que satisfaça todas as cláusulas. O algoritmo para verificar a satisfatibilidade de ϕ é um *procedimento de decisão*, porque dada qualquer fórmula, o algoritmo sempre termina com uma resposta sim/não "correta", após quantidade finita de computações.

Nesse contexto, um solucionador SAT é um algoritmo que toma como entrada uma fórmula ϕ, na forma normal conjuntiva, e decide se ela é satisfazível ou insatisfazível. A fórmula ϕ é dita satisfazível (ou sat) se um solucionador SAT é capaz de encontrar uma interpretação que a torne *verdadeira* (Definição 3.9). A fórmula ϕ é dita insatisfazível (ou *unsat*) se nenhuma interpretação a torna *verdadeira*. No caso satisfazível, os solucionadores SAT podem prover a atribuição às variáveis proposicionais que satisfazem a fórmula ϕ. No caso insatisfazível, quando o solucionador SAT conclui que não há uma atribuição que satisfaz ϕ, seus estados internos podem ser usados para construir uma *prova de resolução* [35].

3.5 Teorias do Módulo da Satisfatibilidade

Um solucionador SMT decide sobre a satisfatibilidade de uma certa fórmula, de primeira ordem, usando diferentes teorias de suporte. Após isso, ele generaliza a satisfatibilidade proposicional suportando funções não interpretadas, aritmética não-linear e linear, vetores de bits, *tuples*, *arrays* e outras teorias de primeira ordem decidíveis.

Definição 3.11. *Dada uma teoria $\mathcal{T}$ e uma fórmula sem quantificação ψ, diz-se que ψ é $\mathcal{T}$-satisfazível se e somente se existir uma estrutura que satisfaça a fórmula e a sentença de $\mathcal{T}$, ou, equivalentemente, se $\mathcal{T} \cup \{\psi\}$ é satisfazível.*

Definição 3.12. *Dado um conjunto $\Gamma \cup \{\psi\}$ de fórmulas de primeira ordem sobre a Σ-teoria, diz-se que ψ é uma $\mathcal{T}$-consequência de Γ e escreve-se $\Gamma \models_{\mathcal{T}} \psi$, se e somente se todo modelo de $\mathcal{T} \cup \Gamma$ é também um modelo de ψ. A verificação de $\Gamma \models_{\mathcal{T}} \psi$ pode ser reduzida de forma usual para a verificação da $\mathcal{T}$-satisfatibilidade de $\Gamma \cup \{\neg\psi\}$.*

A sintaxe das teorias de suporte é sumarizada através de uma notação padronizada (proposta em [36]), quando apropriado, conforme a figura abaixo.

$$
\begin{array}{rcl}
F & ::= & F\ con\ F \mid \neg F \mid A \\
con & ::= & \wedge \mid \vee \mid \oplus \mid \Rightarrow \mid \Leftrightarrow \\
A & ::= & T\ rel\ T \mid Var \mid true \mid false \\
rel & ::= & < \mid \leq \mid > \mid \geq \mid = \mid \neq \\
T & ::= & T\ op\ T \mid \sim T \mid Var \mid Const \\
 & & \mid select(T, i) \mid store(T, i, v) \\
 & & \mid Extract(T, i, j) \mid SignExt(T, k) \mid ZeroExt(T, k) \\
 & & \mid ite(F,\ T,\ T) \\
op & ::= & + \mid - \mid * \mid / \mid rem \mid \texttt{<<} \mid \texttt{>>} \mid \texttt{\&} \mid \texttt{|} \mid \oplus \mid @
\end{array}
$$

Figura 3.9: Sintaxe das Teorias de Suporte

Aqui, F denota as expressões booleanas, T denota os termos construídos sobre inteiros, reais e vetores de bits, enquanto *op* denota os operadores binários. Os conectivos lógicos *con* consistem de conjunção ($\wedge$), disjunção ($\vee$), ou-exclusivo ($\oplus$), implicação ($\Rightarrow$) e equivalência ($\Leftrightarrow$). A interpretação dos operadores relacionais (i.e., $<$, $\leq$, $>$, $\geq$) e dos operadores aritméticos (i.e., $*$, $/$, *rem*) dependem se seus argumentos são vetores de bits com sinal ou sem sinal, inteiros ou números reais. Aqui, o operador *rem* denota o resto com sinal ou sem sinal, dependendo dos argumentos. Os operadores de deslocamento para esquerda e para direita (i.e., `<<`, `>>`) dependem se um vetor de bits sinalizado ou não é usado. Assume-se que o tipo de expressão é claro a partir do contexto. Os operadores bit a bit são e (`&`), ou (`|`), ou-exclusivo ($\oplus$), complemento ($\sim$), deslocamento para direita (`>>`), e deslocamento para esquerda (`<<`). $Extract\,(T, i, j)$ denota a extração do vetor de bits a partir do bit i, até o bit j, para formar um novo vetor de bits de tamanho $i - j + 1$, enquanto o operador @ denota a concatenação de um dado vetor de bits. $SignExt\,(T, k)$ estende o vetor de bits ao vetor de bits sinalizado equivalente, de tamanho $w + k$, onde w é o comprimento original do vetor de bits. Por outro lado, $ZeroExt\,(T, k)$ estende o vetor de bits, com zeros, até o vetor de bits equivalente sem sinal, de tamanho $w + k$. A expressão condicional $ite(f, t_1, t_2)$ toma como primeiro argumento uma fórmula booleana f e, dependendo do seu valor, seleciona o segundo ou o terceiro argumento.

A fim de verificar a satisfatibilidade de uma fórmula, os solucionadores SMT manipulam os termos em uma dada teoria de suporte, usando um procedimento de decisão. Solucionadores SMT, no estado do arte, são construídos sobre solucionadores SAT, para melhorar o desempenho e o suporte a diferentes teorias de decisão [12], [13], [14].

3.6 Verificação Limitada de Modelos Baseada em SMT

A verificação limitada de modelos (BMC), com base em satisfatibilidade booleana (SAT), já é aplicada com sucesso, para verificar software sequencial em sistemas embarcados e descobrir erros sutis em projetos reais [37]. A ideia básica da técnica BMC é verificar (a negação de) uma determinada propriedade, a uma determinada profundidade: dado um sistema de transição M, uma propriedade ϕ e um limite k, o BMC desdobra o sistema k vezes e o traduz para uma condição de verificação ψ, tal que ψ pode ser satisfeita se e somente se ϕ tem um contra-exemplo, a uma profundidade menor ou igual a k. Solucionadores SAT padrão podem ser usados para verificar se ψ é satisfatório. Em BMC de software, o limite k restringe o número de iterações do laço e chamadas recursivas no programa. Sendo assim, o BMC do software gera condições de verificação que refletem o caminho exato em que uma instrução é executada, o contexto em que uma determinada função é chamada e a representação binária precisa das expressões [10]. A prova da validade das condições de verificação, decorrentes de softwares sequenciais (ou *multi-tarefa*), ainda possuem grandes gargalos de desempenho na verificação de software embarcado, apesar das tentativas de lidar com a crescente complexidade dos sistemas, através da aplicação de solucionadores SMT.

Neste trabalho, a ferramenta ESBMC foi utilizada, que é um verificador limitado de modelos para software embarcado, escrito em ANSI-C e baseado em solucionadores de SMT. O ESBMC foi utilizado para verificar os modelos de filtros digitais, implementados na linguagem C. No ESBMC, o problema associado à verificação limitada do modelo é formulado através da construção da fórmula lógica

$$\psi_k = I(s_0) \wedge \bigvee_{i=0}^{k} \bigwedge_{j=0}^{i-1} \gamma(s_j, s_{j+1}). \wedge \neg\phi(s_i) \tag{3.6}$$

Aqui, ϕ é uma propriedade de segurança, I é o conjunto de estados iniciais de M e $\gamma(s_j, s_{j+1})$ é a relação de transição de M, entre os instantes de tempo j e $j+1$. Portanto, $I(s_0) \wedge \bigwedge_{j=0}^{i-1} \gamma(s_j, s_{j+1})$ representa as execuções de M, de tamanho i, e ψ_k pode ser satisfeita se, e somente se, para algum $i \leq k$ existe um estado acessível no instante de tempo i, no qual ϕ é violada. Se ψ_k é satisfazível, então ϕ é violada e o solucionador SMT fornece

uma atribuição satisfatória, a partir da qual é possível extrair os valores das variáveis do programa, para a construção de um contra-exemplo. Um contra-exemplo para uma propriedade ϕ é uma sequência de estados $s_0, s_1, \ldots, s_k$, com $s_0 \in S_0$, $s_k \in S$ e $\gamma\,(s_i, s_{i+1})$, para $0 \leq i < k$. Se ψ_k não é satisfazível, pode-se concluir que nenhum estado de erro é alcançável em k passos ou menos.

O ESBMC suporta completamente o ANSI-C e pode verificar programas sequenciais e *multi-tarefa* que fazem uso de binários, vetores, ponteiros, estruturas, uniões, alocação de memória e aritmética de ponto fixo. Ele pode eficientemente avaliar sobre *underflow* e *overflow* aritmético, segurança ponteiro, vazamentos de memória, violações de limites de matriz, atomicidade e violações de ordem, bloqueios locais e globais, corridas de dados e assertivas especificadas pelo usuário. No ESBMC, diferentes solucionadores SMT podem ser utilizados para se checarem as condições de verificação, além de também poderem ser configurados para usar vetores de bits, codificações em aritmética inteira e aritmética real.

Para exemplificar o processo de verificação, a Figura 3.10(a) mostra um programa escrito em ANSI-C, sintaticamente válido, mas que escreve fora da região de memória alocada para o vetor a, na linha 6. Além disso, na linha 7, há uma assertiva violada, que foi definida pelo usuário, já que o determinado índice da memória não foi inicializado com o valor esperado.

A fim de checar o programa C, a ferramenta de verificação converte o código para a forma de atribuição estática única (single static assignment, SSA), que consiste somente em atribuições condicionais, atribuições não condicionais e assertivas, como mostrado na Figura 3.10(b). Nesse caso, a notação SSA usa a cláusula WITH para representear a operação de armazenamento de dado, no vetor.

Depois dessas transformações, a ferramenta de verificação constrói as fórmulas de restrições C e propriedades P, mostradas respectivamente nas Equações (3.7) e (3.8). Tais fórmulas, sem quantificadores, são construídas utilizando SMT. A fórmula C codifica a primeira parte de ψ_k (ou seja, $I(s_0) \wedge \bigvee_{i=0}^{k} \bigwedge_{j=0}^{i-1} \gamma(s_j, s_{j+1})$) e $\neg P$ codifica a segunda parte (ou seja, $\bigvee_{i=0}^{k} \neg\phi(s_i)$). Por fim, a fórmula $C \wedge \neg P$ é passada para o solucionador SMT, de modo que este cheque a satisfatibilidade.

```
1 int main() {
2   int a[2], i, x;
3   if(x==0)
4     a[i] = 0;
5   else
6     a[i+2] = 1;       //violacao do limite do vetor
7   assert(a[i+1]==1); //violacao da assertiva do usuario
8 }
```

(a)

```
1 g1 = x1 == 0
2 a1 = a0 WITH [i0 := 0]
3 a2 = a0
4 a3 = a2 WITH [2+i0 := 1]
5 a4 = g1 ? a1 : a3
6 t1 = a4[1+i0] == 1
```

(b)

Figura 3.10: (a) Trecho de código em ANSI-C. (b) Instruções SSA para o código em (a).

$$C := \left[\begin{array}{l} g_1 = (x_1 = 0) \\ \wedge\ a_1 = store(a_0, i_0, 0) \\ \wedge\ a_2 = a_0 \\ \wedge\ a_3 = store(a_2, 2 + i_0, 1) \\ \wedge\ a_4 = ite(g_1, a_1, a_3) \end{array} \right] \tag{3.7}$$

$$P := \left[\begin{array}{l} i_0 \geq 0 \wedge i_0 < 2 \\ \wedge\ 2 + i_0 \geq 0 \wedge 2 + i_0 < 2 \\ \wedge\ 1 + i_0 \geq 0 \wedge 1 + i_0 < 2 \\ \wedge\ select\,(a_1, i_0 + 1) = 1 \end{array} \right] \tag{3.8}$$

Aqui, o processo completo de como transformar código C para forma SSA foi omitido e, depois, para as fórmulas sem quantificadores. Para mais detalhes sobre esses processos, trabalhos de Clark et al. [15] e Cordeiro et al. [10] são sugeridos.

3.7 Resumo

Neste capítulo, as estruturas para implementação de filtros digitais foram apresentadas e os diagramas das estruturas na Forma Direta I, Forma Direta II e Forma Direta Transposta foram exibidos, além das formas em cascata e em paralelo, que serão utilizadas como modelos para verificação, neste trabalho. Os efeitos de *overflow* e *wrap-around* oriundos do bloco quantizador, devido ao tamanho limitado da palavra binária para representação numérica, em um filtro em ponto fixo, também foram apresentados. Esses efeitos são configuráveis para cada tipo de verificação abordada nesse trabalho. Também aqui, a representação de um número em ponto fixo no sistema de complemento de dois foi explicada, o que é a forma mais utilizada. Neste trabalho, esse sistema de representação para as implementações dos filtros verificados e considerado. Os fundamentos de lógica foram apresentados e a sintaxe e semântica da lógica proposicional, usada no processo de decisão para a verificação de satisfatibilidade, foi definida. Depois, o conceito da verificação limitada de modelos, utilizando SMT, foi introduzido. O entendimento desses fundamentos é importante para compreender como a ferramenta de verificação descreve o sistema, através da lógica matemática. Por fim, falou-se sobre o verificador de modelos ESBMC, empregado nesse trabalho, e sobre a formulação lógica que este utiliza para verificar as assertivas, que foram geradas a partir de um programa implementado em linguagem ANSI-C. Como resultado, o conteúdo apresentado nesse capítulo fornece todo o embasamento necessário para compreensão do trabalho desenvolvido, que será descrito nas próximas seções.

Capítulo 4

Inspeção Usando Verificador de Modelos Baseado em SMT

Nesse capítulo é apresentada a metodologia para verificação de filtros digitais desenvolvida nesse trabalho de dissertação. A seções a seguir descrevem as etapas propostas para o projeto e verificação do filtro. São apresentados também, os algoritmos que foram desenvolvidos para o sistema de verificação proposto, bem como os aspectos sobre cada propriedade checada pela ferramenta.

4.1 Metodologia de Projeto e Verificação

O projeto de um filtro digital consiste principalmente em definir os coeficientes da equação de diferenças (3.1), ou da função de transferência (3.2), para que o sistema produza a resposta desejada no domínio da frequência. No entanto, a implementação prática requer cuidados quanto à arquitetura do sistema digital, em relação, por exemplo, à representação numérica utilizada, estrutura do filtro a ser embarcado e desempenho. Isso sem referir a outros aspectos que também devem ser considerados, principalmente para a produção em larga escala do sistema como, facilidade de desenvolvimento na plataforma, custo do dispositivo, consumo de energia e espaço ocupado em uma placa de circuito impresso [38]. Esses últimos aspectos, portanto, são os que levam muitas vezes à escolha de sistemas otimizados quanto a capacidade de processamento e comprimento da palavra de dados.

Neste trabalho, as seguintes etapas para o projeto e verificação de um filtro digital são propostas. Primeiro, os parâmetros do filtro são projetados, usando os métodos de preferência (como os apresentados nos livros de Proakis et al. [2] e Oppenheim et al. [31]) ou alguma ferramenta computacional (por exemplo, a ferramenta para projeto de filtros do *toolbox* de processamento de sinais no Matlab [39]). Depois disso, estima-se o intervalo de saída para uma dada entrada, a fim de definir o comprimento da palavra para representar os números em ponto fixo. Uma vez que o comprimento da palavra é definido, os respectivos parâmetros de projeto são introduzidos no modelo do filtro implementado na linguagem C, e em seguida, é possível realizar uma análise de tempo das operações do filtro, considerando uma arquitetura de microprocessador específica. Finalmente, as assertivas (*asserts*) são adicionadas ao modelo, a fim de verificar as propriedades relacionadas à restrição de tempo, *overflow*, ciclo limite, estabilidade, resposta em frequência e erro de saída. Se um *overflow*, um alto erro de saída ou uma violação no domínio da frequência é encontrado, então o comprimento da palavra é aumentado, seguido por uma nova chamada do mecanismo de verificação. Por outro lado, se uma violação de restrição temporal é encontrada, então isso indica que a complexidade do filtro e o comprimento da palavra têm de ser diminuídos, de modo a obter uma melhoria do desempenho. As verificações das posições dos pólos e zeros e do ciclo limite são usadas para afirmar sobre a estabilidade do sistema, que está sujeito aos efeitos de quantização. A Figura 4.1 mostra o fluxo básico da verificação de um projeto de filtro.

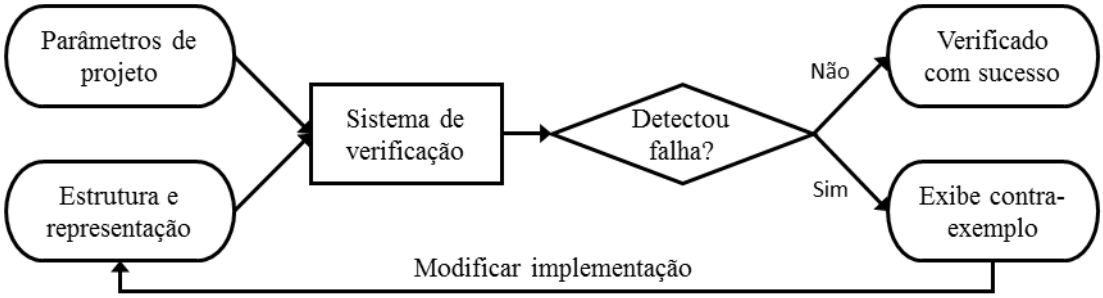


Figura 4.1: Fluxo de projeto e verificação.

Deve ser notado que, embora as verificações de resposta em frequência, posições dos pólos e zeros, e restrição temporal são realizadas dentro do contexto do ESBMC, essas propriedades não exploram a verificação exaustiva através de entradas não-determinísticas. De qualquer modo, são muito importantes para serem incluídas no conjunto de ferra-

mentas, uma vez que há um compromisso em estabelecer a conformidade de todas as propriedades mencionadas e definir uma adequada representação em ponto fixo.

Para possibilitar a utilização do ESBMC para o propósito da verificação de filtros em ponto fixo, foi necessário criar bibliotecas e funções úteis para a modelagem desses sistemas. As seções a seguir descrevem sobre os algoritmos implementados e sobre o método de verificação de cada propriedade abordada nesse trabalho.

4.2 Biblioteca para Aritmética de Ponto Fixo

A codificação de aritmética de ponto flutuante em um *framework* BMC leva a grandes fórmulas e, consequentemente, um elevado consumo de memória e tempo de verificação. Diante disso, os mecanismos de verificação típicos suportam a representação de ponto fixo, usando vetor de bits e aritmética real. Em relação à aritmética de vetor de bits, que foi aplicada neste trabalho, ela assume que a parte inteira e fracionária do número têm a mesma largura de bits antes e depois do ponto de raiz. Assim, para uma variável *double* de 64 bits, 32 bits são usados para representar a parte fracionária e os 32 bits restantes são usados para representar a parte inteira, incluindo o sinal. No entanto, este trabalho busca a verificação de sistemas com variadas reapresentações do ponto fixo e com diferentes quantidade de bits para a parte inteira e parte fracionária. Sendo assim foi implementada uma biblioteca que inclui definições e funções para operações com o tipo de variável de ponto fixo definido a partir de um inteiro de 32 bits, conforme a Figura 4.2.

```
1  typedef int32_t fxp32_t;
```

Figura 4.2: Tipo de variável para representação de ponto fixo.

A Tabela 4.1 mostra a assinatura de algumas das principais funções utilizadas para aritmética de ponto fixo.

A biblioteca também possibilita a configuração de alguns parâmetros para a aritmética de ponto fixo através das seguintes definições, exibidas na Tabela 4.2 a seguir.

As funções da biblioteca foram utilizadas para implementação dos modelos dos filtros nesse trabalho. Em cada caso de teste para avaliação do método, as definições das quan-

Tabela 4.1: Funções para aritmética de ponto fixo

Função	Descrição
fxp32_t fxp_wrap(int64_t a, fxp32_t l fxp32_t u)	Envolve o parâmetro a dentro do intervalo ente l e u
fxp32_t fxp_get_int_part(fxp32_t a)	Retorna a parte inteira do parâmetro a
fxp32_t fxp_get_frac_part(fxp32_t a)	Retorna a parte fracionária do parâmetro a
fxp32_t fxp_quant(int64_t a)	Aplica saturação, wraparound ou detecta overflow em a
fxp32_t fxp_add(fxp32_t a, fxp32_t b)	Retorna a adição de a e b
fxp32_t fxp_sub(fxp32_t a, fxp32_t b)	Retorna da subtração de a e b
fxp32_t fxp_mult(fxp32_t a, fxp32_t b)	Retorna da multiplicação de a e b
fxp32_t fxp_int_to_fxp(int a)	Converte o valor do inteiro a para ponto fixo
int fxp_to_int(fxp32_t a)	Converte o valor de ponto fixo para inteiro
fxp32_t fxp_float_to_fxp(float a)	Converte o valor de ponto flutuante para ponto fixo
float fxp_to_float(fxp32_t a)	Converte o valor para representação de ponto flutuante

Tabela 4.2: Definições para aritmética de ponto fixo

Definição	Descrição	Valor
FXP_WIDTH	Define a largura total da palavra de dados	32
FXP_PRECISION	Define o número de bits da parte fracionária	De 0 a 16
FXP_IWIDTH	Define o número de bits da parte inteira com sinal	De 1 a 16
OVERFLOW_MODE	Define o procedimento em caso de *overflow*	1 = detectar *overflow* 2 = saturar 3 = *wrap-around*

tidades de bits para parte inteira e parte fracionária foram configuradas de acordo com a representação em ponto fixo desejada.

4.3 Algoritmo para Filtros Digitais

A Figura 4.3 apresenta um gráfico de fluxo de dados de um filtro genérico na Forma Direta I. Os blocos de adição e multiplicação também incluem o efeito de quantização sobre o valor do resultado, que considera a representação de ponto fixo usado para o sistema. O quantizador pode ser configurado para saturação, *wrap-around* ou lançamento de erro, quando o resultado de uma operação excede os limites representáveis. A implementação do algoritmo do filtro em linguagem C pode ser feita de diversas maneiras. Para esse

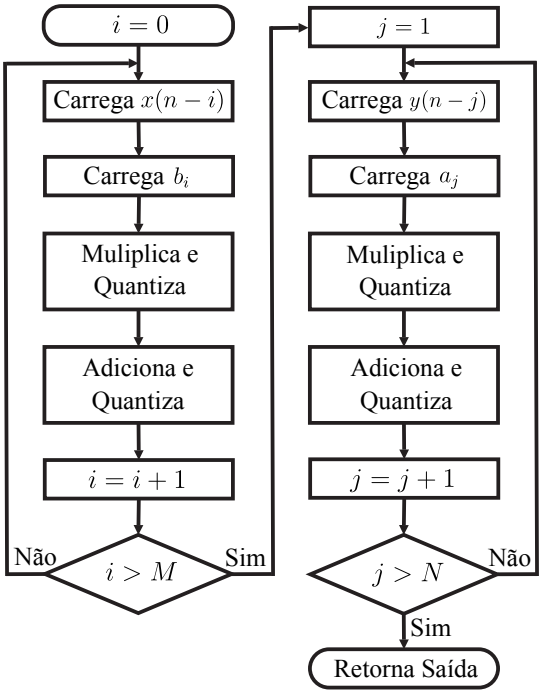


Figura 4.3: Fluxo lógico de filtro na Forma Direta I.

trabalho o trecho de código exibido na Figura 4.4 é a função utilizada como modelo do filtro IIR na forma direta I em ponto fixo. É possível perceber que a sequência lógica desse

```
1  fxp32_t iirOutFixed(fxp32_t y[], fxp32_t x[], fxp32_t a[],
2                      fxp32_t b[], int N, int M) {
3    fxp32_t *a_ptr, *y_ptr, *b_ptr, *x_ptr;
4    fxp32_t sum = 0;
5    a_ptr = &a[1];
6    y_ptr = &y[N - 1];
7    b_ptr = &b[0];
8    x_ptr = &x[M - 1];
9    int i, j;
10
11   for (i = 0; i < M; i++) {
12     sum = fxp_add(sum, fxp_mult(*b_ptr++, *x_ptr--));
13   }
14   for (j = 1; j < N; j++) {
15     sum = fxp_sub(sum, fxp_mult(*a_ptr++, *y_ptr--));
16   }
17   return sum;
18 }
```

Figura 4.4: Código da função do filtro IIR na Forma Direta I.

código é diretamente resultante da implementação da estrutura da Figura 3.3. Além dessa

função, foram também implementados, similarmente, os algoritmos para as estruturas de filtros na Forma Direta II e Forma Direta Transposta II, em linguagem C e com uso da biblioteca de ponto fixo.

Antes de abordar sobre os métodos de verificação, aqui deve ser destacado que as funções de filtragem não são utilizadas para as checagens de resposta em frequência e de estabilidade, uma vez que o procedimento de verificação para essas propriedades considera apenas o efeito de quantização nos coeficientes do filtro e como a diferente localização dos pólos e zeros afeta o sistema.

4.4 Verificação de Overflow

No projeto de um filtro em ponto fixo, é preciso especificar o número de bits para representação da parte inteira e para a parte fracionária dos valores numéricos. Primeiramente, a faixa de saída do filtro, para um determinado sinal de entrada, deve ser estimada. Tal procedimento baseia-se tipicamente em abordagens analíticas ou em simulações. Assim, o projetista deve especificar um comprimento de palavra adequado para representar as variáveis, considerando-se também os erros de quantização na resposta do sistema. Vários autores têm proposto técnicas para encontrar o comprimento da palavra para os coeficientes de filtros digitais [40, 41]. No entanto, não é garantido que as variáveis do sistema estarão dentro do intervalo esperado, independentemente da sequência de entrada.

Neste trabalho, as assertivas são codificadas no bloco quantizador e a máquina de verificação é configurada para usar entradas não-determinísticas no intervalo especificado, a fim de detectar *overflows* no filtro digital, para um determinado comprimento de palavra de ponto fixo. Para qualquer resultado de adição ou de multiplicação, durante a operação do filtro, se existir um valor que exceda o intervalo representável, uma instrução *assert* o detecta como uma violação. Sendo assim, um literal $l_{overflow}$ é gerado, com o objetivo de representar a validade de cada operação de adição e multiplicação, de acordo com a seguinte restrição:

$$l_{overflow} \Leftrightarrow (MIN \leq FP) \wedge (FP \leq MAX), \tag{4.1}$$

onde FP é a aproximação do valor em ponto fixo para o resultado dos somadores e multiplicadores, MIN e MAX são os valores mínimo e máximo representáveis pelo formato

binário do ponto fixo, respectivamente (como descrito anteriormente na Seção 3.3)

Como exemplo ilustrativo, suponha um sistema com único pólo descrito pela equação de diferença:

$$y(n) = -a\ y(n-1) + x(n). \tag{4.2}$$

Esse é um sistema estável de entrada-limitada/saída-limitada (BIBO, *bounded-input bounded-output*), no qual a saída é limitada em amplitude por:

$$|y(n)| \leq x_{max} \sum_{k=-\infty}^{\infty} |h_k|\,, \tag{4.3}$$

onde x_{max} é o máximo valor de saída, e h_k é a resposta ao impulso do sistema. Para a Equação (4.2), com $a = -1/2$, pode ser mostrado que o somatório da norma da resposta ao impulso converge para 2, utilizando série geométrica. Considerando, para este exemplo'em particular, uma entrada no intervalo $[-1; 1]$, a saída estará portanto, no intervalo $[-2; 2]$ (isto é, simplesmente multiplicar o intervalo de entrada por $\sum |h_k|$). Tendo isto para a implementação, um projetista poderia, de forma otimista, escolher para representar o número em ponto fixo com 2 bits para a parte inteira, incluindo o sinal e 4 bits para a parte fracionária. O intervalo resultante para este formato é $[-2; 1,9375]$, com um erro de $\pm 0,03125$.

O método proposto pode ser utilizado para verificar a configuração proposta. Os coeficientes da Equação (4.2) são usados para o modelo de filtro em linguagem C, com a representação numérica definida anteriormente. Se a máquina de verificação é executada, tendo em conta a gama de entradas entre $[-1; 1]$, então ele mostra um contra-exemplo em que o sistema sofre um *overflow* para uma determinada sequência de entradas. Por exemplo, pode ser facilmente demonstrado que uma sequência de entradas $x = \{1, 1, 1, 1, 1, 1\}$ conduz a um *overflow* na saída, conforme mostrado na Tabela 4.3.

Tabela 4.3: Exemplo de *overflow* em um filtro em ponto fixo.

n	**1**	**2**	**3**	**4**	**5**	**6**
$x(n)$	1	1	1	1	1	1
$y(n)$	1,0000	1,5000	1,7500	1,8750	1,9375	1,96875

Para esse caso em particular, o *overflow* poderia ser facilmente previsto pela análise da soma da resposta ao impulso, ou pela simulação utilizando uma entrada degrau constante.

Entretanto, para sistemas de ordem mais alta, pode ser difícil avaliar precisamente a resposta ao impulso infinita, ou encontrar uma sequência de entrada que provoque uma falha, como foi também observado por Cox et al. [9]. E ainda, a soma da resposta ao impulso é útil para inferir sobre os limites de saída do sistema, mas não para as operações intermediárias.

Considerando os casos onde as operações intermediárias podem causar *overflow*, o filtro definido por

$$y(n) = 0,703125\ y(n-1) - 0,5\ y(n-2) + \\ 0,75\ x(n) - 0,703125\ x(n-1) + 0,75\ x(n-2) \tag{4.4}$$

pode ilustrar tais situações. Nesse exemplo, é considerado que os valores em ponto fixo são representados com 2 bits para a parte inteira e 6 bits para a parte fracionária. O somatório $\sum |h_k|$ para esse filtro converge para $1,8279$. Sendo assim, se entradas entre $[-1;1]$ forem aplicadas ao sistema, então as saídas serão representáveis dentro do intervalo $[-2;1,984375]$ em ponto fixo. Quando é feita a verificação BMC desse sistema, com implementação na Forma Direta I e Forma Direta Transposta II, o sistema de verificação proposto exibe um contra-exemplo que causa um *overflow* em uma operação intermediária. A Figura 4.5 ilustra o *overflow* no soma após o primeiro estágio da estrutura na Forma Direta I, quando a entrada do contra-exemplo é aplicada ao filtro. Para o filtro na Forma Direta II, a verificação termina com sucesso, sem emitir qualquer falha de *overflow*, devido a diferente ordem de execução das operações nesse caso particular. Como consequência, tais observações então reforçam a aplicação do BMC para filtros digitais, como uma ferramenta de teste auxiliar.

4.5 Verificação de Ciclo Limite

Em um filtro estável ideal, a saída deve aproximar assintoticamente de um nível de estado estacionário determinado pela função de transferência do filtro [42]. Entretanto, se um problema de ciclo limite existir, ele pode se manifestar como uma oscilação estável ou como um nível diferente de zero na saída, mesmo para uma entrada de nível zero. Este efeito é causado pelos arredondamentos da quantização ou *overflows* durante a operação do filtro.

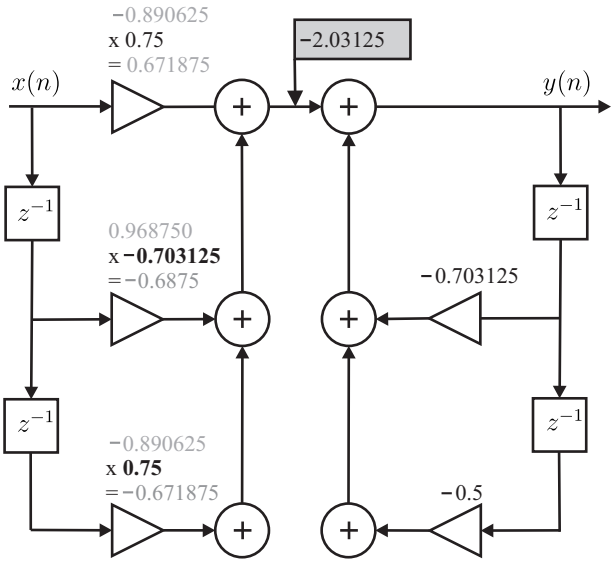


Figura 4.5: *Overflow* em um filtro na Forma Direta I.

Para verificar a presença de ciclo limite em uma implementação específica de um filtro em ponto fixo, a rotina do bloco quantizador é configurada através de uma variável, para realizar o *wrap-around* quando acontecer um *overflow*. O comportamento esperado será como mostrado na Figura 3.7, o que significa que o verificador não deverá detectar as falhas de *overflow* como no caso anterior, durante a verificação de ciclo limite. Além disso, o sistema é configurado para aplicar uma entrada nula no filtro e um estado inicial não-determinístico para as saídas anteriores. Então a execução do filtro é desdobrada para um número limitado de entradas. Uma assertiva é adicionada para detectar uma falha caso um conjunto de estados anteriores das saídas se repetir para a resposta à entrada nula. Pode-se notar que este método é diferente do que foi apresentado por Cox et al. [9], que visa encontrar um ciclo limite, comparando uma janela da saída com outra janela da saída dentro de um número limitado de instantes posteriores.

Como exemplo, o mesmo sistema descrito pela equação de diferença (4.2) é considerado. Aqui, o sistema também é modelado usando 2 bits para a parte inteira e 4 bits para a parte fracionária, mas agora é definido um sinal de entrada nulo. Se executar o sistema de verificação para o modelo implementado, então ele encontra uma condição inicial particular que leva o sistema para um ciclo limite. Na Tabela 4.4 é listada a resposta do sistema para essa condição particular. As colunas y_2 e y_{10} representam a resposta do filtro em formato binário e decimal, respectivamente. Devido ao arredondamento da parte

fracionária do número em ponto fixo, considerando uma condição inicial $y(-1) = 0,125$, é possível ver na Tabela 4.4 que para $a = 0,5$ a saída começa a se repetir depois $n = 2$. Da mesma forma, para $a = -0,5$, a saída se mantém em um valor de estado estacionário não nulo, em vez de decair para zero.

Tabela 4.4: Ciclo Limite para um filtro de único polo.

$a = 0,5_{10} = 0,1000_2$			$a = -0,5_{10} = 1,1000_2$		
n	y_2	y_{10}	n	y_2	y_{10}
-1	0,0010	0,125	-1	0,0010	0,125
0	1,0001	-0,0625	0	0,0001	0,0625
1	0,0001	0,0625	1	0,0001	0,0625
2	1,0001	-0,0625	2	0,0001	0,0625
3	0,0001	0,0625	3	0,0001	0,0625

4.6 Verificação de Resposta em Frequência

O projeto de um filtro é, como dito, realizado de modo a definir os coeficientes para uma dada resposta em frequência desejada. No entanto, as alterações nos coeficientes devido à quantização, em ponto fixo, geralmente modificam as respostas em magnitude e fase do filtro [25], como mostrado na Figura 4.6. Nessa figura a curva tracejada é a resposta em magnitude do sistema projetado, já a curva sólida é a resposta em magnitude resultante para implementação em ponto fixo usando 1 bit de sinal, 7 bits inteiros e 6 bits para a parte fracionária [43].

Na presente abordagem, a conformidade da resposta em frequência também é analisada com o uso dos coeficientes do filtro projetado em representação de ponto flutuante. São consideradas as propriedades do filtro de acordo com as condições adotadas (por exemplo, banda de passagem, frequência de corte, banda rejeição e ganhos em cada região de frequência), e o número de bits no utilizados para a representação de ponto fixo.

Dado que N é o numero de pontos a ser verificado na Transformada Discreta de Fourier, $h[n]$ é definido como a resposta ao impulso do filtro, e $H(k)$ é o equivalente no domínio da frequência. Dessa forma,

$$H(k) = \sum_{n=0}^{N-1} h(n) e^{-j(2\pi/N)kn}. \tag{4.5}$$

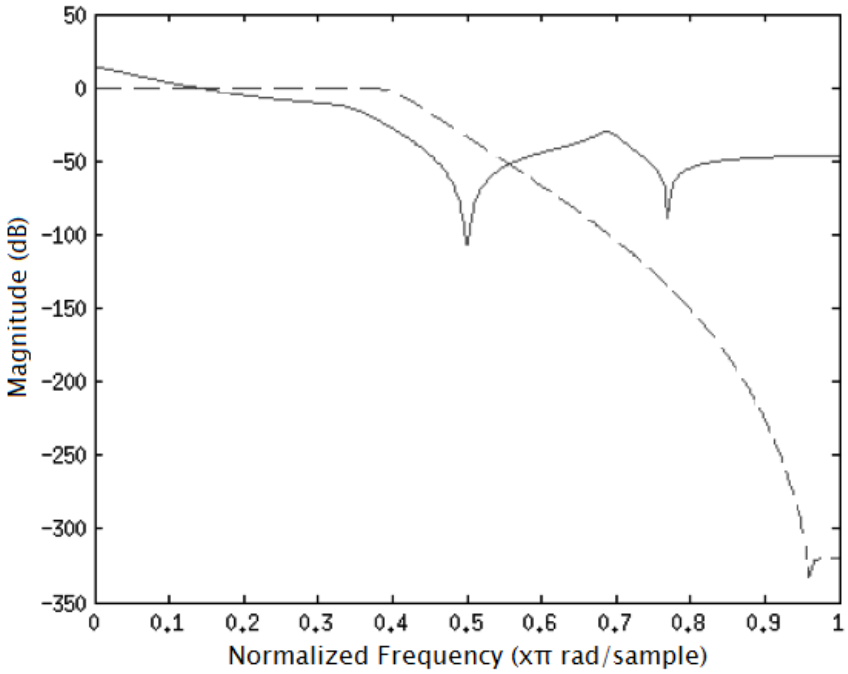


Figura 4.6: Resposta em magnitude de um filtro IIR de ordem 12, em ponto flutuante (curva sólida) e em ponto fixo $< 8, 6 >$ (curva tracejada).

Os valores complexos de $H(k)$ são obtidos da expansão numérica da Equação (4.5), dados os coeficientes do filtro. Então, o valore absoluto de $|H(k)|$ é usado para a verificação, em comparação com resposta em frequência especificada, em cada frequência $\omega_k = 2\pi k/N$.

Com a resposta $H(k)$, o algoritmo de verificação busca pela violação das seguinte expressão, para determinar se o sistema atende às especificações de resposta em magnitude, quando implementado em ponto fixo. O literal l_{mag} é codificado com a restrição

$$\begin{aligned} l_{mag} \Leftrightarrow ((|H(k)| > A(k)) \wedge (\omega_1 \leq \frac{2\pi k}{N} \leq \omega_2) \wedge P(k)) \\ \vee ((|H(k)| < A(k)) \wedge (\omega_1 \leq \frac{2\pi k}{N} \leq \omega_2) \wedge \neg P(k)), \end{aligned} \tag{4.6}$$

onde $A(k)$ é o valor aceitável de magnitude para a faixa passante ou faixa de rejeição, e $P(k)$ indica o perfil que deve ser 1 para faixa passante ou 0 para faixa de rejeição, para cada frequência ω_k. Se a expressão apresentada é violada, um erro é lançado, o que indica que o número de bits não é suficiente para as dadas restrições do projeto. Aqui deve ser destacado que o valor aceitável relativo a variação da magnitude deve ser definido pelo projetista de acordo com a aplicação. Nesse trabalho, conforme está descrito adiante, na Seção 5 de avaliação experimental, em todos os casos foi estabelecida uma margem

aceitável de variação de 5% sobre o ganho absoluto do sistema projetado. Esse valor foi definido simplesmente para avaliar o funcionamento da assertiva de verificação da resposta em magnitude.

4.7 Verificação de Pólos e Zeros

No sistema proposto, a estabilidade também pode ser verificada com base nos pólos e zeros do sistema. Tal processo consiste em encontrar os pólos da função de transferência de sistemas causais (como definido na Seção 3.1), e então verificar se o valor absoluto é inferior a um. Existem diversos métodos descritos na literatura, que podem ser usados para avaliar a estabilidade de sistemas, como o mostrado por Diniz et al. [30], baseado em divisões polinomiais. Nesse trabalho particularmente, a biblioteca *Eigen* [44], que possui bom desempenho e confiabilidade, é usada para encontrar as raízes dos polinômios determinados pelos coeficientes do filtro.

O algoritmo primeiramente define a matriz companheira, tal que, dado um polinômio $p(t) = t^n + a_{n-1}t^{n-1} + \cdots + a_1 t + a_0$, a matriz companheira é definida por

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -a_0 & -a_1 & -a_2 & \cdots & -a_{n-1} \end{bmatrix}.$$

Então, a matriz A é reduzida à forma real de Schur [45], e finalmente, a decomposição de Schur é aplicada para calcular os autovalores, que são as raízes do polinômio [46].

A complexidade do algoritmo é $O(n^3)$. E o numero máximo de iterações é igual ao numero de colunas na matriz companheira multiplicado por 40, uma vez que são geradas raízes com alta precisão (i.e., 10^{-12}) para a verificação do filtro. Tal processo é realizado tanto para os polinômios compostos pelos pólos quanto pelos zeros, com o objetivo de cancelar as raízes que tenham mesmo valor.

Finalmente, é verificado se cada pólo tem valor absoluto menor que um, o que determina se o sistema é estável, caso contrário o sistema é instável [31]. Aqui, o processo de

verificação simplesmente procura a violação do seguinte literal

$$l_{stable} \Leftrightarrow \lambda_1 < 1 \ \wedge \ \lambda_2 < 1 \ \wedge \ ... \ \wedge \ \lambda_n < 1 \tag{4.7}$$

onde os λ's são os valores são os valores absolutos dos pólos do sistema extraídos a partir da matriz resultante, exceto para aqueles cancelados.

O processo descrito é realizado pelo sistema de verificação através da chamada da função *__ESBMC_check_stability(float *a, float *b)* e os coeficientes quantizados devem ser previamente declarados em matrizes em ponto flutuante a e b no arquivo fonte que é verificado pelo ESBMC. O breve trecho de programa na Figura 4.7 descreve um exemplo com a chamada da função para checar a estabilidade.

```
1  float a[] = { 1.0f, 0.34375f, 0.328125f };
2  float b[] = { 0.25f, -0.484375f, 0.25f };
3  int main(void) {
4    assert(__ESBMC_check_stability(a, b));
5    return 0;
6  }
```

Figura 4.7: Trecho de código para verificação de estabilidade.

4.8 Verificação de Restrição Temporal

Existem estruturas eficientes para a implementação de filtros, como a forma em malha (Lattice) e os métodos de filtragem com base na transformada rápida de Fourier [30]. Estes métodos visam reduzir o número de operações aritméticas e o custo computacional. No entanto, os métodos de convolução no domínio do tempo com base em formas diretas ainda são predominantes, tanto em implementações em hardware como em software, devido à sua simplicidade. Em aplicações em tempo real, um filtro pode receber dados na mesma taxa em que os processa e envia à saída. Sendo assim, a verificação das restrições de tempo torna-se necessária, especialmente em filtros de ordem elevada, que apresentam mais operações aritméticas e maiores atrasos de grupo.

Na abordagem proposta, o modelo de filtro implementado em C é novamente utilizado, mas agora para verificar e avaliar sobre o tempo máximo aceitável para as operações do filtro. Como um exemplo, uma função de um filtro IIR foi implementada e compilada

```
1    sum += *b_ptr++ * *x_ptr--;
```

(a)

```
1    MOV.W  @r9+,r12  5 cycles
2    MOV.W  @r9+,r13  5 cycles
3    SUB.W  # 4,r10   5 cycles
4    MOV.W  4(r10),r14  3 cycles
5    MOV.W  6(r10),r15  3 cycles
6    CALL # __fs_mpy  5 cycles
7    MOV.W  r7,r14    1 cycle
8    MOV.W  r8,r15    1 cycle
9    CALL # __fs_add  5 cycles
10   MOV.W  r12,r7    1 cycle
11   MOV.W  r13,r8    1 cycle
```

(b)

Figura 4.8: (a) Trecho de código de uma implementação de filtro digital. (b) Instruções *assembly* geradas para o trecho de código em (a).

para executar em um MSP430G2231, que é um microcontrolador de potência ultra-baixa baseado em um CPU RISC de 16-bits [47]. Aqui neste caso é considerado um processador dedicado à tarefa de filtragem simplificando a análise no que diz respeito a efeitos de *caching*, ordem de execução e *pipelining* [48]. Uma vez que a implementação do filtro em tal arquitetura é direta, assume-se aqui que o comportamento temporal das operações é repetível e previsível. Sendo assim, tendo o arquivo *assembly* gerado a partir da compilação, este pode ser comparado com o código fonte de modo a identificar as instruções para cada segmento do programa. Em seguida, uma análise do pior caso de tempo de execução (WCET) pode ser realizada na função do filtro, considerando o número de ciclos necessários para executar as instruções e as iterações. Kim et al. [17] descreve um método que utiliza a análise estática e verificação de modelos para inspecionar sobre o tempo de trechos de programas, semelhantemente ao que é feito aqui.

Por exemplo, o trecho de código mostrado na Figura 4.8(a) é usado para realizar a multiplicação das entradas anteriores com os coeficientes b_k em 3.1. A Figura 4.8(b) mostra o código da Figurz 4.8(a) convertido em instruções *assembly*, usando o compilador CCS v4 [49].

Pode-se então perceber que cada instrução leva um número diferente de ciclos de *clock*

para executar, e com base nessa informação, é possível calcular o número de ciclos que serão necessários para cada operação. Para o MSP430G2231, a frequência interna é de até 16 MHz, o que dá uma duração de ciclo de $62,5$ ns. Uma vez que o tempo total de processamento de instruções associadas está disponível, então isso pode ser usado para incrementar uma variável de temporização e adicionar uma instrução *assert* para detectar qualquer violação da restrições temporal.

O valor de restrição pode ser facilmente estimado, com base na taxa de amostragem do sistema, por exemplo, se ele opera usando uma taxa de amostragem de 48 KHz (que é comumente usado em sistemas digitais de áudio). Então isso significa que, após cada janela de $20,8\mu$s, novos dados são obtidos a partir da entrada do sistema, e o filtro tem a função de processar a saída dentro deste tempo. Formalmente, um literal l_{timing} é gerado para representar a validade do tempo de resposta, com uma restrição

$$l_{timing} \Leftrightarrow ((N \times T) \leq D), \tag{4.8}$$

onde N é o número de ciclos consumidos pelo filtro, T é o tempo de ciclo e D é o prazo para processar.

4.9 Verificação de Erro

Como já mencionado e amplamente conhecido, o cálculo usando palavra de comprimento finito leva a erros de arredondamento e truncamento [25, 31]. Neste trabalho, as discrepâncias presentes na saída do filtro, devido ao arredondamento dos coeficientes e resultados de operações aritméticas, são consideradas. Como mostrado na Figura 3.7, o erro de quantificação E, devido ao arredondamento de um número representado com l bits de precisão, é

$$-2^{-l-1} \leq E \leq 2^{-l-1}. \tag{4.9}$$

Note que a precisão no cálculo de filtros IIR e FIR é limitada pelo comprimento da palavra especificado na realização do sistema digital. O efeito do arredondamento dos coeficientes, muda as posições dos pólos e zeros do sistema e modifica a resposta em frequência, como indicado nas Seções 4.6 e 4.7. Tais modificações, em conjunto com o arredondamento de resultados das operações, causam variações na saída do filtro, que podem também ser observadas no domínio do tempo.

O uso de variáveis de ponto flutuante proporciona uma melhor aproximação para representação de um número real do que uma representação de ponto fixo com o mesmo número de bits, uma vez que abrange uma faixa dinâmica maior, variando a resolução ao longo do intervalo. No entanto, implementações práticas de filtros digitais são tipicamente realizadas com representação de ponto fixo. Tendo isso, aqui é proposta a verificação se os erros de saída de um filtro estão dentro de uma faixa aceitável (definida pelo projetista). Para esta finalidade, a saída do filtro implementado com representação de ponto fixo com comprimento de palavra reduzido é comparado com a saída de uma referência na mesma estrutura, implementada utilizando variáveis de precisão dupla. É importante notar que ambos os modelos apresentam erros de quantização. No entanto, uma vez que nos modelos de referência são utilizadas variáveis *double* completas, a amplitude do erro é muito menor do que nos modelos de ponto fixo com precisão reduzida. Assim, tendo em conta que o número de bits de precisão l_d nos modelos de referência são maiores do que nos modelos projetados, o erro de quantificação E_r de valores e coeficientes resultantes da redução de bits de precisão é dado por

$$-2^{-l-1} + 2^{-l_d-1} \leq E_r \leq 2^{-l-1} - 2^{-l_d-1}. \tag{4.10}$$

A expressão anterior mostra que o valor de erro calculado usando o método proposto é afetado pela precisão do modelo de referência. No sistema implementado, foram utilizadas variáveis de ponto fixo de 64 bits para o modelo de referência, com 32 bits de precisão. Os experimentos foram executados para modelos com menos de 16 bits para a parte fracionária, proporcionando assim uma boa aproximação nos cálculos de erro.

Para os modelos de ponto fixo verificados, os valores calculados são saturados para o número máximo representável em caso de *overflow*. A mesma entrada é aplicada a ambos os modelos, sendo que o vetor de entrada utiliza valores não determinísticos do conjunto $\{2^{k-1} - 2^{-l}, 2^{-l}, 0, -2^{-l}, 2^{k-1}\}$, isto é, os valores para a amplitude máxima e mínima do sinal de entrada. Assim, os erros devidos a quantização e saturação são explorados.

Durante a operação da função do filtro, o erro acumulado pode aumentar para além dos limites do intervalo do erro de quantização. O seguinte literal é gerado e verificado para determinar se o erro de saída está de acordo com uma margem aceitável:

$$l_{error} \Leftrightarrow |y - y_d| \leq M\ 2^{-l-1}, \tag{4.11}$$

onde y é a saída do sistema projetado, y_d é a saída do sistema utilizando precisão dupla, e $M\ 2^{-l-1}$ é a margem de tolerância utilizada. O processo de verificação procura a negação desse literal. Quando um contra-exemplo é encontrado, ele indica que o erro de saída é maior do que o desejável naquela estrutura de filtro implementada.

4.10 Resumo

Neste capítulo foi apresentada a metodologia utilizada para verificação de potenciais problemas na implementação de filtros digitais em ponto fixo. Para a verificação de *overflow* utiliza-se uma assertiva que verifica se algum resultado está dentro do intervalo representável pela palavra binária. Para verificação do ciclo limite a afirmação utilizada observa se há alguma repetição dos estados do filtro na condição de entrada nula. No caso da verificação da restrição temporal, é feita uma análise estática a partir das durações dos ciclos de instrução do filtro implementado em C e é verificado se o tempo de resposta atende o *deadline* estabelecido a partir da frequência de amostragem do sistema. Já na verificação de resposta em frequência, o valor da magnitude do sistema em ponto fixo é comparado com a especificação de projeto. Para verificação de estabilidade, é checado o valor dos pólos, que devem estar dentro do círculo unitário. Por fim, na verificação do erro de saída, é checado se a diferença entre a saída do sistema quantizado e de um sistema com maior precisão está dentro de um limiar determinado. Todas essas verificações são executadas dentro do contexto da ferramenta ESBMC. Logo, nesse trabalho são checadas mais propriedades do que em trabalhos anteriores, dentro de uma ferramenta de verificação integrada. A Tabela 4.5 a seguir resume as métricas utilizadas para verificação de cada propriedade.

Tabela 4.5: Resumo das métricas de verificação

Verificação	**Métrica**
Overflow	Algum valor menor que -2^k ou maior que $2^k - 2^l$
Ciclo Limite	Valores dos estados de y iguais aos valores iniciais
Magnitude	Valor de $\|H(k)\|$ é 5% maior na faixa de rejeição ou 5% menor na faixa passante
Estabilidade	Valor absoluto dos pólos maior ou igual a 1
Tempo	Pior caso de tempo de execução da filtragem maior que período de amostragem
Erro	Valor absoluto do erro de saída maior do que $2^{-l-1} - 2^{-l_d-1}$

Capítulo 5

Avaliação Experimental

Este capítulo é dividido em três partes. A Seção 5.1 restringe o escopo dos experimentos feitos e reforça sobre as propriedades suportadas pelo sistema de verificação proposto, que foram aqui avaliadas. A configuração experimental é descrita na Seção 5.2, enquanto a Seção 5.3 apresenta os resultados da verificação de alguns modelos de filtros digitais, utilizando a abordagem proposta.

5.1 Escopo dos Experimentos

No presente trabalho, a abordagem proposta não é comparada com a apresentada por Cox et al. [9]. Na verdade, as conclusões encontradas naquele trabalho, em relação ao uso do recurso de precisão de bits para encontrar problemas em filtros, também foram aplicados aqui. Além disso, Cox et al. implementou uma ferramenta em linguagem OCaml com chamadas diretas à API do solucionador Z3 SMT, que tende a ser mais rápido em encontrar *overflows* e ciclos limites, uma vez que existem poucos passos de análise para produzir o modelo do sistema a ser verificado.

A metodologia proposta, por sua vez, verifica o código C efetivo de filtros digitais que se destinam a ser incorporados em micro-controladores e DSPs, o que é muito mais perto de implementações reais, onde as construções específicas de C (por exemplo, a aritmética de ponteiro e comparações) são usados para implementar a Equação (3.1), e desta forma torna as condições de verificação mais difíceis. Além disso, o presente trabalho se diferencia em explorar outras técnicas de verificação, que são usadas não só

para detecção de *overflows* e ciclos limites, mas também para checar estabilidade e desvios no domínio da frequência e do tempo.

Vale destacar que o presente trabalho também propõe um conjunto de verificações necessárias, através da exploração de um verificador de modelos no estado da arte, com o objetivo de ajudar a escolher a representação do sistema e estrutura, respeitando as especificações do projeto.

5.2 Configuração Experimental

Na Tabela 5.1, estão descritos alguns filtros escolhidos com diferentes tipos de projeto, o número de coeficientes de realimentação N, o número de coeficientes diretos M, intervalo do sinal de entrada, e comprimento de palavra. Note que a coluna *Bits* indica o comprimento da palavra binária para a parte inteira e fracionária dos números em ponto fixo, incluindo o bit de sinal. Note ainda que o comprimento da palavra para a representação em ponto fixo é estimado com base no valor do somatório $\sum |h_k|$ e a faixa de entrada, a fim de obter filtros otimizados em termos de redução do número de bits. A coluna *In* mostra o número de entradas consecutivas que são aplicadas ao filtro. Para cada entrada aplicada, a função do filtro é executada a fim de calcular a saída, portanto o número na coluna *In* representa a quantidade limite de desdobramentos do programa (isto é, a função de filtro).

Na Tabela 5.1, os filtros de 1 a 11 são verificados em três estruturas diferentes: Forma Direta I (DFI), Forma Direta II (DFII) e Forma Direta Transposta II (TDFII), de modo a investigar como as diferentes realizações do filtro podem interferir na ocorrência de falhas. As verificações de resposta em frequência e estabilidade não são executadas para diferentes estruturas, uma vez que são considerados apenas os efeitos de quantização sobre os coeficientes do filtro. Muitos dos casos de teste selecionados usam estruturas de segunda ordem, uma vez que tal realização é amplamente utilizada e pode ser aplicada como blocos para formar sistemas de ordem superior, conforme referido na Seção 3.2.

Durante os experimentos de resposta em frequência, o foco foi verificar a resposta em magnitude, através da fixação de uma margem aceitável de 5% em relação ao ganho absoluto da referência projetada em ponto flutuante. A verificação de fase não é abordada

Tabela 5.1: Seleção de Filtros Digitais Avaliados

#	**Filtro**	N	M	$\sum \lvert h_k \rvert$	**Entrada**	**Bits**	**In**
1	LP-IIR	2	1	2	$[-1;1]$	$<2;4>$	6
2	LP-BW-IIR	3	3	1,2	$[-1,6;1,6]$	$<2;5>$	6
3	LP-IIR	3	1	4	$[-1;1]$	$<3;4>$	6
4	LP-IIR	3	1	1,56	$[-1;1]$	$<2;4>$	6
5	HP-CSI-IIR	3	3	1,33	$[-1;1]$	$<2;6>$	6
6	BP-Elliptic-IIR	3	3	1,24	$[-1;1]$	$<2;8>$	6
7	BS-BW-IIR	3	3	1,81	$[-1,1;1,1]$	$<2;8>$	6
8	BP-Elliptic-IIR	5	5	0,91	$[-1,1;1,1]$	$<1;7>$	10
9	HP-BW-IIR	5	5	1,58	$[-1,27;1,27]$	$<2;6>$	10
10	BP-CSI-IIR	5	5	1,51	$[-1,33;1,33]$	$<2;6>$	10
11	HP-Elliptic-IIR	7	7	5,39	$[-1;1]$	$<3;11>$	14
12	HP-Casc-IIR	6	6	12,4	$[-1;1]$	$<5;5>$	14
13	BS-Casc-IIR	9	9	2,45	$[-1;1]$	$<3;5>$	19
14	LP-Paral-IIR	6	6	16,6	$[-1;1]$	$<5;5>$	13
15	LP-Casc-IIR	6	6	7,64	$[-1;1]$	$<4;4>$	13
16	MB-Paral-IIR	9	9	2,75	$[-1;1]$	$<4;4>$	19
17	BP-Casc-IIR	6	6	1,39	$[-1;1]$	$<3;7>$	13
18	LP-FIR	1	31	1,94	$[-1;1]$	$<2;6>$	31
19	MB-FIR	1	9	2,18	$[-1;1]$	$<2;6>$	13
20	MB-FIR	1	25	1,47	$[-1;1]$	$<2;8>$	25
21	LP-WiFi-FIR	1	21	2,92	$[-1;1]$	$<3;5>$	21

aqui, pois é sabido que, para filtros FIR de fase linear (que, diferentemente dos IIR, a preservação das características da fase são um aspecto bastante importante a ser considerado), a quantização dos coeficientes não afeta a linearidade da resposta em fase [2].

Em relação aos experimentos de verificação de erro, é coerente definir um intervalo que seja maior que o intervalo do erro de quantização, definido pela Equação (4.9). Sendo assim, uma margem aceitável de duas vezes a precisão foi fixada, ou seja, de -2^{-l+1} a 2^{-l+1}.

Para a avaliação das restrições temporais, em todos os casos foram consideradas

as condições de operação de um processador de 16 MHz em um sistema com taxa de amostragem igual a 48 KHz. Tal taxa foi adotada devido a sua utilização como frequência de amostragem de áudio padrão, comumente empregada em equipamentos profissionais. Note que a taxa de amostragem do sistema não interfere nas outras condições de verificação como *overflow* e ciclo limite, uma vez que estas são apenas consequências da aritmética de ponto fixo.

Os resultados para filtros FIR, a respeito das verificações de *overflow* e ciclo limite, não são apresentados aqui, uma vez que esses problemas podem ser facilmente evitados neste tipo de sistema. O *overflow* pode ser evitado através da aplicação de critérios conservadores com base na soma do módulo da resposta ao impulso, para determinar a palavra de comprimento. Além disso, problemas de ciclo limite ocorrem somente em sistemas com realimentação. De fato, os sistemas FIR checados para essas propriedades tiveram a verificação concluída com êxito ou excederam o tempo limite sem encontrar um único contra-exemplo.

Aqui, o ESBMC v1.21 [1] foi empregado e configurado para usar o solucionador SMT Z3 v3.2 [12], com a aritmética de vetor de bits habilitada, uma vez que essa produz menos alarmes falsos que a aritmética inteira ou real (como também observado por Cox et al. [9]).

Para cada caso de teste, o mecanismo de verificação foi invocado configurando o nome do arquivo com as definições dos teste, o tempo limite e o solucionador utilizado. [2] Note que são desativadas as assertivas para detecção de violação de limites de vetores, segurança de ponteiros e divisão por zero, já que o presente trabalho é focado em verificar especificamente as propriedades relacionadas ao filtro, conforme descrito anteriormente no Capítulo 4. A chamada ESBMC acima é, assim, utilizado para verificar as propriedades de segurança relacionadas a *overflow* aritmético, quando `<file>` referir, por exemplo, a *verify_overflow.c*. A fim de verificar a existência de ciclo limite, restrições de tempo, erro de quantização, resposta em frequência e estabilidade, os seguintes arquivos podem ser referenciados pela chamada ESBMC acima, respectivamente: *verify_limitcycle.c*, *verify_timing.c*, *verify_error.c*, *verify_freqz.c* e *verify_stability.c*. Cada arquivo contém as chamadas de filtro necessárias e assertivas usadas para verificação de cada propriedade.

[1] ESBMC está disponível em *www.esbmc.org*, juntamente com os *benchmarks* para que outros pesquisadores possam reproduzir os resultados

[2] Particularmente, a ferramenta ESBMC foi chamado como se segue: `esbmc <file> --no-bounds-check --no-pointer-check --no-div-by-zero-check --timeout` *1h* `--z3-bv`

Todos os experimentos foram realizados em um processador Intel Core i7-2600, 3.40 GHz com 24 GB de memória RAM rodando no sistema operacional Linux Fedora 64-bits. Para todos os casos de teste, o limite de tempo de verificação foi definido para 3600 segundos, com exceção dos sistemas em paralelo e em cascata e para os filtros FIR, em que o tempo limite é de 7200 segundos. Os tempos decorridos apresentados foram medidos usando o comando *time*.

5.3 Resultados Experimentais

Depois de selecionar os filtros digitais, seus parâmetros são usados como entradas para o modelo de filtro codificado em linguagem C. As Tabelas 5.2, 5.3 e 5.4 resumem os resultados obtidos para os filtros verificados usando o ESBMC. Elas mostram o resultado como *V* ou *F*, ou seja, se a verificação foi concluída com sucesso ou não, respectivamente. O tempo de verificação também é mostrado para cada tipo de propriedade, e, quando uma verificação ultrapassa o limite de tempo, o caso de teste é marcado como *timeout*, representado por *TO*.

Os resultados mostram que o sistema proposto pode detectar falhas em vários tipos de filtros digitais, que são implementados com diferentes estruturas, ordens ou formatos de ponto fixo. No entanto, o tempo de verificação tende a ser maior para os filtros de alta ordem e formatos com longo comprimento de palavra, uma vez que estes levam a condições de verificação maiores e mais difíceis. Em particular, o tempo de verificação tende a ser mais alto para a verificação de *overflow* em filtros que contêm um elevado número de coeficientes diretos e de realimentação, como pode ser visto a partir do caso de teste 9 diante. Observe também que o sistema atingiu o tempo limite durante a verificação de ciclos limites, com filtros digitais que apresentam comprimento de palavra longo para a representação da parte fracionária (por exemplo, caso de teste 11). A verificação de erro também leva a tempos de processamento elevados, dado que a saída precisa ser calculada duas vezes, isto é, para a referência de precisão mais elevada e para o projetado com comprimento de palavra reduzido. Verificações de resposta em frequência e estabilidade não usam entradas não-determinísticas, no entanto, eles contêm longos desdobramentos em cálculos numéricos, que incorrem em tempos de processamento consideráveis. Ademais,

Tabela 5.2: Resumo dos resultados para os filtros IIR verificados

#	Filtro	Tipo	Overflow		Ciclo Limite		Magnitude		Estabilidade		Temporização		Erro	
			Saída	**Tempo**	**Saída**	**Tempo**	**Saída**	**Tempo**	**Saída**	**Tempo**	**Saída**	**Tempo**	**Saída**	**Tempo**
1	LP-IIR	DFI	F	3	F	8	V	9	V	<1	V	1	V	6
		DFII	F	2	F	13					V	<1	V	4
		TDFII	F	2	F	8					V	<1	V	4
2	LP-BW-IIR	DFI	F	2	V	417	V	22	V	<1	F	<1	V	31
		DFII	F	1	V	709					F	1	F	5
		TDFII	V	30	F	447					F	<1	V	32
3	LP-IIR	DFI	V	10	F	21	V	11	V	<1	V	<1	V	14
		DFII	V	12	F	55					V	1	V	10
		TDFII	V	39	F	135					V	<1	V	12
4	LP-IIR	DFI	V	4	V	88	V	17	V	<1	V	<1	V	11
		DFII	V	5	V	106					V	<1	V	9
		TDFII	V	11	F	101					V	<1	V	10
5	HP-CSI-IIR	DFI	V	10	V	941	F	20	V	<1	F	<1	V	48
		DFII	V	27	V	1776					F	1	V	57
		TDFII	V	14	F	331					F	<1	V	49
6	BP-Ellip-IIR	DFI	V	9	F	37	V	115	V	<1	F	1	V	53
		DFII	F	2	F	70					F	<1	F	16
		TDFII	V	11	F	105					F	1	V	46
7	BS-BW-IIR	DFI	F	3	F	437	V	24	V	<1	F	<1	F	17
		DFII	F	2	F	112					F	<1	F	11
		TDFII	V	117	F	321					F	<1	V	73
8	BP-Ellip-IIR	DFI	F	3	V	8	F	67	F	<1	F	1	F	168
		DFII	F	2	V	14					F	1	F	79
		TDFII	F	2	V	12					F	<1	F	147
9	HP-BW-IIR	DFI	F	15	F	1060	F	75	V	<1	F	1	F	125
		DFII	F	4	F	1506					F	1	F	145
		TDFII	TO	3600	F	1485					F	<1	TO	3600
10	BP-CSI-IIR	DFI	V	96	TO	3600	F	89	V	<1	F	1	F	255
		DFII	F	11	F	2395					F	1	F	106
		TDFII	V	139	F	2092					F	<1	F	238
11	HP-Ellip-IIR	DFI	F	12	TO	3600	F	288	F	<1	F	1	TO	3600
		DFII	F	4	TO	3600					F	1	TO	3600
		TDFII	F	5	TO	3600					F	1	TO	3600

as restrições de tempo são facilmente verificadas, uma vez que este procedimento consiste em apenas verificar o tempo de resposta de um trecho de código sequencial.

Outra observação importante sobre os tempos de verificação, como observado durante os testes realizados para a validação da metodologia proposta, é que os casos de teste com falha tendem a ser verificados mais rapidamente. A principal razão é que o algoritmo de verificação de modelos interrompe um procedimento de verificação sempre que encontra

um contra-exemplo. No entanto, casos em que nenhum defeito é encontrado tendem a ter tempos de verificação superiores, chegando até mesmo a alcançar o tempo limite de teste. Em tais cenários, um amplo conjunto de entradas é aplicado, o que gera um grande número de condições de verificação e torna o procedimento ser muito demorado. Isso pode ser notado, por exemplo, no processo de verificação de *overflow* para os casos de teste 6, 7 e 10, e também durante a verificação de ciclo limite no caso de teste 5. Na verificação de erro no caso de teste 9, o resultado indica falha para as implementações na Forma Direta I e Forma Direta II, em menos de 150 segundos; para a Forma Direta Transposta II, a verificação chega ao tempo limite sem encontrar um valor de erro além dos limites fixados. Isto indica que é difícil encontrar uma entrada que produz um valor elevado de erro para esta estrutura particular, o que sugere que tal realização pode operar adequadamente na maioria das vezes, no entanto, não pode ser garantido, uma vez que a verificação do modelo não foi concluída.

A verificação de tempo relatou falhas para todos os casos com ordem superior a 2, que apresentam mais de 3 coeficientes diretos ou de realimentação. Na verdade, considerando as estruturas modeladas implementadas em C, apenas os filtros 1, 3, 4 atendem aos requisitos de restrição temporal, durante a execução na plataforma especificada.

Pode ser visto, especialmente nas verificações de *overflow* e de ciclo limite, que um filtro pode falhar ou passar de acordo com a sua estrutura de implementação. Isso ocorre devido à ordem de execução das operações intermediárias, que mudam de uma estrutura para outra. A verificação da restrição temporal também é afetada pela estrutura do filtro, uma vez que o número de adições e multiplicações são diferentes em cada forma. O caso de teste 4, por exemplo, não deve ser implementado utilizando a Forma Direta Transposta II, a fim de evitar a ocorrência de oscilações de ciclo limite. Quanto aos casos de teste 6 e 7, no entanto, não há nenhuma opção viável, a menos que o projetista use alguma técnica (por exemplo, saturação) para evitar o ciclo limite.

Por exemplo, no caso de teste 2, a verificação detecta falhas de *overflow* para a estrutura na Forma Direta I e Forma Direta II, além de uma falha de ciclo limite para a estrutura na Forma Direta Transposta II; as demais propriedades forma concluídas com êxito. Nesta caso, o projetista que pretende implementar este filtro na Forma Direta I ou II precisaria utilizar um acumulador com um maior número de bits para a parte inteira

da representação numérica ou alterar a escala para evitar o *overflow*. Quando se utiliza a Forma Direta Transposta II, o projetista poderia modificar o modo de *overflow* do sistema, a fim de realizar uma aritmética de saturação para evitar as oscilações de ciclo limite. As técnicas para evitar *overflow* e ciclo limite, como referido por Proakis et al. em [2], geralmente aumentam o ruído no filtro digital. Sendo assim, após a realização de tais modificações, o projetista deve necessariamente executar a verificação novamente, a fim de assegurar que o erro de saída está dentro de uma margem aceitável.

Tabela 5.3: Resumo dos resultados para os filtros em cascata e em paralelo

			Overflow		Ciclo Limite		Magnitude		Estabilidade		Temporização		Erro	
#	**Filtro**	**Tipo**	**Saída**	**Tempo**	**Saída**	**Tempo**	**Saída**	**Tempo**	**Saída**	**Tempo**	**Saída**	**Tempo**	**Saída**	**Tempo**
12	HP-Casc-IIR	TDFII	TO	7200	TO	7200	V	34	V	<1	F	2	F	1584
13	BS-Casc-IIR	DFII	F	259	TO	7200	F	44	V	<1	F	7	TO	7200
14	LP-Paral-IIR	DFII	V	911	F	3656	V	33	V	<1	F	3	TO	7200
15	LP-Casc-IIR	DFII	F	261	F	4059	V	34	V	<1	F	3	F	189
16	MB-Paral-IIR	TDFII	TO	7200	F	6668	V	38	V	<1	F	<1	TO	7200
17	BP-Casc-IIR	TDFII	TO	7200	TO	7200	F	38	F	<1	F	<1	TO	7200

A Tabela 5.3 descreve os resultados obtidos para filtros implementados em cascata e em paralelo. Estes sistemas utilizam blocos de segunda ordem na suas estruturas, como mencionado na Seção 3.2. A maior parte dos resultados mostra que o número de bits usados para essas implementações não é suficiente para evitar problemas causados pelo comprimento reduzido da palavra, encontrando *overflow* e violações de erro nos casos apresentados. Aqui, os procedimentos de verificação para erro, *overflow* e ciclo limite são muito demorados, devido à maior ordem de tais sistemas. Sendo assim, uma alternativa para os os casos de teste que atingiu o tempo limite é, pelo menos, garantir a conformidade dos blocos de segunda ordem separadamente, uma vez que essas estruturas podem ser verificadas mais rapidamente, como mostrado na Tabela 5.2.

Pode ser destacado ainda, na Tabela 5.3, o caso de teste 17 que apresenta falha na verificação de estabilidade. O filtro passa faixa nesse exemplo foi projetado de forma a obter uma faixa de passagem bem estreita, tornando os pólos do sistema bem próximos ao círculo de raio unitário. Em casos assim, o efeito de quantização dos coeficientes pode fazer com que o módulo do valor do pólo se torne maior ou igual a 1. Portanto, filtros que

operam próximos ao limite de estabilidade, como filtros *notch* e filtros de pico, estão mais sujeitos a falhas de estabilidade quando implementados em ponto fixo, e estas podem ser detectadas através do método proposto.

Tabela 5.4: Resumo dos resultados para os filtros FIR verificados

			Magnitude		Estabilidade		Temporização		Erro	
#	**Filtro**	**Tipo**	**Saída**	**Tempo**	**Saída**	**Tempo**	**Saída**	**Tempo**	**Saída**	**Tempo**
17	LP-FIR	DFI	V	61	V	<1	F	1	TO	7200
18	MB-FIR	DFI	V	61	V	<1	F	1	V	7915
19	MB-FIR	DFI	F	58	V	<1	F	1	TO	7200
20	LP-WiFi-FIR	DFI	F	49	V	<1	F	1	TO	7200

Os resultados coletados a partir dos experimentos de filtros FIR são descritos na Tabela 5.4. Os sistemas de alta ordem causam tempos de verificação elevados, devido à busca de um contra-exemplo que produza um alto erro de saída. As verificações de magnitude, estabilidade e restrição temporal também são rapidamente realizadas, como nos outros casos apresentados. Os filtros multi-banda 19 e 20 também puderam ser verificados para essas propriedades. Como pode ser visto, no caso de teste 20, a falha de magnitude indica que o número de bits de precisão não são suficientes para cumprir as especificações.

5.4 Resumo

Neste capítulo foi apresentada a forma como foram configurados e executados os experimentos para avaliação do método proposto nesse trabalho. Foi abordado sobre a seleção dos *benchmarks* bem como a chamada do ESBMC para verificação dos filtros implementados em linguagem C. Por fim, foram exibidas tabelas com os resultados obtidos nos testes feitos. As tabelas exibem os tipos de falhas, quando encontradas, em cada filtro, demonstrando a eficácia do método para verificação de cada propriedade nos *benchmarks* selecionados. Os resultados apresentados mostram que a metodologia proposta abrange um conjunto de verificações exigidas comumente nos projetos de filtros digitais, a fim de definir o formato de representação de ponto fixo. Além disso, é também útil para determinar a estrutura para a realização do sistema, isto é, observando se a implementação do

sistema é praticável, tendo em conta os restrições do projeto. A importância na verificação do conjunto de propriedades se dá devido à necessidade em se estabelecer o compromisso entre as restrições conflitantes, como mencionado na análise dos casos de teste.

Capítulo 6

Conclusão

Neste trabalho foi abordado sobre a verificação de problemas em implementações de filtros digitais que utilizam representação numérica em ponto fixo. A metodologia proposta busca verificar a corretude das implementações através de verificação formal. Para isso é utilizado o ESBMC, um verificador limitado de modelos para software escritos em linguagem ANSI-C, baseado em teorias do módulo da satisfatibilidade. Sendo assim, diferentes modelos de filtros foram definidos em linguagem C e utilizando uma biblioteca para aritmética de ponto fixo, que permite avaliar estruturas com diferentes comprimentos da palavra de dados.

O método proposto permite que o projetista verifique formalmente uma determinada implementação em uma estrutura específica, além de ajudar a definir o formato do ponto fixo para representar adequadamente os dados numéricos. Em particular, o sistema de verificação auxilia o projetista a detectar problemas causados pelo comprimento finito da palavra, como estouro, ciclos limite, desvio da resposta em frequência, estabilidade e erro na saída, em filtros digitais. Os resultados experimentais mostram que as falhas podem ser detectadas em filtros digitais de ordem baixa e média, com a largura de bits arbitrária. No entanto, a verificação de filtros de ordem elevada, com maior comprimento de palavra tende a ser um problema difícil, que demanda alto tempo de verificação, devido à grande exploração de espaço de estado.

Adicionalmente, existe uma contribuição com um novo método, baseado na análise WCET, em conjunto com o BMC, o qual é utilizado para verificar restrições temporais em filtros digitais. Uma vez que os modelos de filtros digitais foram implementados em

linguagem C, o método proposto pode também ser aplicado com outras ferramentas de BMC existentes, tirando partido da sua robustez e eficiência.

O método de verificação proposto nesse trabalho pode ser realizado através de um sistema unificado, capaz de verificar mais propriedades do que em trabalhos anteriores, permitindo avaliar mais aspectos relativos à implementação de filtros digitais em ponto fixo. Com isto, é possível confrontar os resultados das verificações de propriedades cujas restrições são conflitantes. Por exemplo, aumentar o número de bits fracionários, em detrimento dos bits inteiros, pode diminuir o erro de saída e o desvio da resposta em magnitude, porém isso pode causar *overflow* durante as operações intermediárias.

Os resultados mostram que em determinados sistemas, a simples mudança na forma de implementação da estrutura pode evitar falhas, já que a ordem de execução das operações é diferente em cada forma. Portanto, o conjunto de ferramentas desenvolvido aqui pode ajudar projetistas de sistemas a definir a representação e estrutura adequada, a fim de atender os requisitos funcionais e de desempenho.

6.1 Trabalhos Futuros

A modelagem matemática de controladores digitais, que são sistemas tipicamente tratados na teoria de controle moderno, é muito semelhante à dos filtros digitais abordados nesse trabalho. Portanto a metodologia proposta aqui poderia ser estendida para verificação de controladores, tratando aspectos específicos desses sistemas aplicados tanto em malha aberta quanto em malha fechada.

Outro tema de pesquisa bastante promissor está relacionado à localização de falhas em programas. Griesmayer et al. [50], apresentaram uma abordagem automática para localização de falhas em programas em C. O método é baseado em verificação de modelos e é capaz de corrigir o programa a partir do contra-exemplo gerado. Em trabalhos futuros, essa técnica também pode ser estendida para executar a localização automática de problemas na implementação de filtros digitais. O contra-exemplo gerado na verificação de um sistema com falha, pode ser usado para correção dos parâmetros e estrutura do filtro, de modo a atender os requisitos de projeto.

Referências Bibliográficas

[1] 754-2008, I. *Ieee standard for floating-point arithmetic*. IEEE Computer Society, 2002.

[2] J. G. Proakis and D. G. Manolakis. *Digital signal processing: Principles, algorithms and applications*. Prentice Hall, 1996.

[3] S. R. Parker and S. F. Hess. Limit-cycle oscillations in digital filters. *IEEE Transactions on Circuit Theory*, v. 18, n. 6, p. 687–697, 1971.

[4] T. A. C. M. Claasen, W. F. G. Mecklenbrauker and J. B. H. Peek. Effects of quantization and overflow in recursive digital filters. *IEEE Transactions on Acoustics, Speech and Signal Processing*, v. assf-24, n. 6, p. 517–529, 1976.

[5] N. Henzel. Digital filter design with constraints in time and frequency domains. *Proc. of the 4th International Conference on Computer Recognition Systems*, v. 30, p. 169–176, 2005.

[6] SYNOPSYS®. *Spw*. Disponível em http://www.synopsys.com/Systems/BlockDesign/DigitalSignalProcessing/Pages/Signal-Processing.aspx. Última visita no dia 14 de Fevereiro, 2014.

[7] MathWorks®. *Simulink fixed point*. Disponível em http://www.mathworks.com/products/simfixed. Última visita no dia 14 de Fevereiro, 2014.

[8] W. Sung and Ki-Il Kum. Simulation-based word-length optimization method for fixed-point digital signal processing systems. *IEEE Transactions on Signal Processing*, v. 43, n. 12, p. 3087–3090, 1995.

[9] A. Cox, S. Sankaranarayanan and Bor-Yuh E. Chang. A bit too precise? Bounded verification of quantized digital filters. *TACAS*, , n. 7214, p. 33–47, 2012.

[10] L. Cordeiro, B. Fischer and J. Marques-Silva. SMT-based bounded model checking for embedded ANSI-C software. *IEEE Transactions on Software Engineering*, v. 38, n. 4, p. 957–974, 2012.

[11] A. Burns and A. Wellings. *Real-time systems and programming languages*. Addison Wesley Longmain, 2009.

[12] L. M. de Moura and N. Bjørner. Z3: An efficient smt solver. *TACAS*, v. LNCS 4963, p. 337–340, 2008.

[13] R. Brummayer and A. Biere. Boolector: An efficient smt solver for bit-vectors and arrays. *TACAS*, v. LNCS 5505, p. 174–177, 2009.

[14] C. Barrett and C. Tinelli. CVC3. *Proceedings of the 19th International Conference on Computer Aided Verification*, v. LNCS 4590, p. 298–302, 2007.

[15] E. Clarke, D. Kroening and F. Lerda. A tool for checking ANSI-C programs. *TACAS*, , n. 2988, p. 168–176, 2004.

[16] C. Baier and J. P. Katoen. *Principles of model checking (representation and mind series)*. The MIT Press, 2008.

[17] S. Kim, H. D. Patel and S. A. Edwards. Using a model checker to determine worst-case execution time. Technical report, Computer Science Department. Columbia University, 2009.

[18] R. Barreto, L. Cordeiro and B. Fischer. Verifying embedded c software with timing constraints using an untimed model checker. *8th Brazilian Workshop on Real-time Systems*, p. 46–52, 2011.

[19] C. Weinstein and A. V. Oppenheim. A comparison of roundoff noise in floating point and fixed point digital filter realizations. *Proceedings of the IEEE*, v. 57, n. 6, p. 1181–1183, 1969.

[20] A. V. Oppenheim and C. Weinstein. Effects of finite register length in digital filtering and the fast fourier transform. *Proceedings of the IEEE*, v. 60, n. 8, p. 957–976, 1972.

[21] P. H. Bauer and L. J. Leclerc. 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.

[22] K. Premaratne, E. C. Kulasekere, P. H. Bauer and L. J. Leclerc. An exhaustive search algorithm for checking limit cycle behavior of digital filters. *IEEE Transactions on Signal Processing*, v. 44, n. 10, p. 2405–2412, 1997.

[23] D. A. Bailey and A. A. Beex. Simulation of filter structures for fixedpoint implementation. *Proceedings of the Twenty-Eighth Southeastern Symposium on System Theory*, p. 270–274, 1996.

[24] E. Abdel-Raheem and F. El-Guibaly. A tool for two's complement, bit-level, fixed-point simulation of digital filters. *The 7th IEEE International Conference on Electronics, Circuits and Systems, 2000*, v. 1, p. 587–590, 2000.

[25] Behzad Akbarpour and Sofiène Tahar. Error analysis of digital filters using HOL theorem proving. *Journal of Applied Logic*, v. 5, n. 4, p. 651–666, 2007.

[26] A. Cox, S. Sankaranarayanan and Bor-Yuh E. Chang. A bit too precise? verification of digital filters. *Software Tools for Technology Transfer 2013*, 2013.

[27] INRIA. *Ocaml: The caml language*. Disponível em http://caml.inria.fr/. Última visita no dia 5 de Fevereiro, 2014.

[28] Microsoft Research. *Z3: An efficient theorem prover*. Disponível em http://z3.codeplex.com. Última visita no dia 5 de Fevereiro, 2014.

[29] C. F. Fang, R. A. Rutenbar and Tsuhan Chen. Fast, accurate static analysis for fixed-point finite-precision effects in dsp designs. *ICCAD*, p. 275–282, 2003.

[30] P. S. R. Diniz, E. A. B. da Silva and S. L. Netto. *Digital signal processing: System analysis and design*. Cambridge University Press, 2010.

[31] A. V. Oppenheim , R. W. Schafer and J. R. Buck. *Discrete-time signal processing.* Prentice Hall, 1999.

[32] A. R. Bradley and Z. Manna. *The calculus of computation: Decision procedures with applications to verification.* Springer-Verlag New York, Inc., 2007.

[33] M. Huth and M. Ryan. *Logic in computer science: modelling and reasoning about systems.* Cambridge University Press, 2004.

[34] L. Cordeiro. *SMT-based bounded model checking of multi-threaded software in embedded systems.* PhD thesis - University of Southampton, 2011.

[35] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, , n. 12, p. 23–41, 1965.

[36] SMT-LIB. *The satisfiability modulo theories library.* Disponível em http://combination.cs.uiowa.edu/smtlib. Última visita no dia 4 de Abril, 2009.

[37] A. Biere. *Bounded model checking.* Handbook of Satisfiability, 1999.

[38] Analog Devices. *Fixed-point vs floating-point digital signal processing.* Disponível em http://www.analog.com/en/content/fixed-point_vs_floating-point_dsp/fca.html. Última visita no dia 04 de Abril, 2014.

[39] Signal Processing Toolbox. *Filter design and analysis tool (fdatool).* Disponível em http://www.mathworks.com/ help/dsp/ref/fdatool.html. Última visita no dia 2 de Fevereiro, 2014.

[40] E. Avenhaus. On the design of digital filters with coefficients of limited word length. *IEEE Trans. Audio Electroacoust*, v. AU-20, p. 206–212, 1972.

[41] C. Charalambous and M. J. Best. Optimization of recursive digital filters with finite word length. *IEEE Trans. Acoust., Speech, Signal Processing*, v. ASSP-22, p. 424–431, 1974.

[42] T. Brubaker and J. Gowdy. Limit cycles in digital filters. *IEEE Transactions on Automatic Control*, v. 17, p. 675–677, 1972.

[43] M. L. Freitas, M. Y. R. Gadelha, L. Cordeiro, W. S. S. Junior and E. B. L. Filho. Verificação de propriedades de filtros digitais implementados com aritmética de ponto fixo. *XXXI Simpósio Brasileiro de Telecomunicações*, 2013.

[44] B. JANSSENS, W. B.; LIMAM, K. *Building finite-element matrix expressions with boost proto and the eigen library.* C++ now 2013, Aspen, United States, 2013.

[45] GUENNEBAUD, G. *Eigen: a c++ linear algebra library.* First Plafrim Scientific Day, Bourdeaux, 2011.

[46] ARBENZ, P.; KRESSNER, D. *Lectures notes on solving large scale eigenvalue problems.* Zurich, 2014.

[47] Texas Instrument. *Msp430g2231 mixed signal controler.* Disponível em http://www.ti.com/lit/ds/symlink/msp430g2231-ep.pdf. Última visita no dia 5 de Fevereiro, 2014.

[48] D. A. Patterson and J. L. Hennessy. *Computer organization and design - the hardware / software interface, (revised 4th edition).* The Morgan Kaufmann Series, Academic Press, 2012.

[49] Texas Instrument. *Code composer studio integrated development environment for msp430.* Disponível em http://www.ti.com/tool/ccstudio-msp430. Última visita no dia 5 de Fevereiro, 2014.

[50] A. Griesmayer, S. Staber, and R. Bloem. Automated fault localization for c programs. *Journal Electronic Notes in Theoretical Computer Science (ENTCS)*, v. 174, n. 4, p. 95–111, 2007.

Apêndice A

Publicações

A.1 Referente à Pesquisa

- **R. B. Abreu**, L. C. Cordeiro e E. B. L. Filho, "Verifying Fixed-Point Digital Filters using SMT-Based Bounded Model Checking", *XXXI Simpósio Brasileiro de Telecomunicações*, 2013

- **R. B. Abreu**, M. Y. Ramalho, L. C. Cordeiro, E. B. L. Filho e W. S. Sabino, "Verifying Fixed-Point Digital Filters using SMT-Based Bounded Model Checking", *IEEE Transactions on Computers*, 2014 (submetido)

A.2 Contribuições em outras Pesquisas

- I. V. Bessa, **R. B. Abreu**, J. E. C. Filho e L. C. Cordeiro, "SMT-Based Bounded Model Checking of Fixed-Point Digital Controllers", *40th Annual Conference of IEEE Industrial Electronics Society*, IECON, 2014