Verificação de Propriedades de Filtros Digitais Implementados com Aritmética de Ponto Fixo

Mauro L. de Freitas, Mikhail Y. R. Gadelha, Lucas Cordeiro, Waldir S. S. Júnior e Eddie B. L. Filho

***Resumo*— Na área de processamento digital de sinais, uma das principais tarefas executadas é o projeto de filtros digitais. Atualmente, tal procedimento é realizado com a ajuda de ferramentas computacionais, que geralmente supõem coeficientes de filtros representados com aritmética em ponto flutuante. Entretanto, durante a fase de implementação, que geralmente ocorre em processadores digitais de sinais ou matrizes de portas de campo programáveis, a representação dos coeficientes obtidos pode ser realizada em aritmética de ponto fixo ou inteira, o que muitas vezes resulta em comportamentos inesperados ou até mesmo em filtros instáveis. O presente trabalho aborda esse problema e propõe uma metodologia de avaliação baseada em verificadores do tipo *efficient SMT-based context-bounded model checker* [2], [3], com o objetivo de analisar se a quantidade de bits utilizada, na representação dos coeficientes, resultará em um filtro com as mesmas características especificadas na etapa de projeto. Simulações mostram que erros relativos à resposta em frequência e *overflow* são passíveis de identificação com a metodologia proposta, o que proporciona maior confiabilidade ao projeto.**

***Palavras-Chave*— ESBMC, *Model checking*, Filtros digitais, Resposta em frequência, *overflow*.**

***Abstract*— In the digital signal processing area, one of the most important tasks is the digital filter design. Currently, this procedure is performed with the aid of computational tools, which generally assume filter coeficients represented with floating-point arithmetic. However, during the implementation phase, which is often done in digital signal processors or field programmable gate arrays, the representation of the obtained coefficients can be carried out through integer or fixed-point arithmetic, which often results in unexpected behavior or even unstable filters. The present work addresses this issue and proposes an evaluation methodology based on the efficient SMT-based context-bounded model checker approach [2], [3], in order to analyze if the number of bits used in the coeficients representation will result in a filter with the same features specified in the design phase. Simulations show that errors regarding frequency response and overflow are likely to be identified with the proposed methodology, which provides greater reliability to the project.**

***Keywords*— ESBMC, Model checking, Digital Filters, Frequency Response, overflow.**

I. INTRODUÇÃO

Filtros digitais com resposta ao impulso finita (*Finite Impulse Response* - FIR) ou infinita (*Infite Impulse Response* - IIR) são utilizados em diferentes áreas, como processamento digital de sinais (PDS), sistemas de controle, telecomunicações, instrumentação médica e eletrônica de consumo. Em geral, tais aplicações variam de simples filtros seletores de frequência e filtros adaptativos até equalizadores e bancos de filtros, cujo objetivo é modificar as características de um dado sinal, de acordo com requisitos preestabelecidos.

O projeto de filtros digitais segue uma rica teoria matemática, tanto no domínio da frequência quanto no domínio do tempo, e é geralmente realizado através de ferramentas como o MATLAB [1], que normalmente assumem precisão em ponto flutuante. Entretanto, pode haver uma grande disparidade entre o projeto de filtros e a sua implementação prática. Por exemplo, muitos projetos são implementados em processadores digitais de sinais (*Digital Signal Processor* - DSP) ou matrizes de portas de campo programáveis (*Field Programmable Gate Arrays* - FPGA), que podem apresentar precisão finita, baseada em aritmética de ponto fixo (com custo e complexidade mais baixos), ao passo que o projeto geralmente utiliza precisão em ponto flutuante.

Essa diferença tem o potencial de gerar efeitos indesejados na resposta em frequência do filtro, tanto em magnitude quanto em fase, além de problemas como *overflow* e instabilidade, que são indesejados. Tal comportamento é devido a erros de quantização causados pela precisão finita, o que resulta em coeficientes diferentes dos projetados. Como resultado, surgem dúvidas, na fase de implementação, quanto à eficácia do filtro digital e a quantidade de bits necessária para a sua representação, de modo que os parâmetros de projeto ainda sejam satisfeitos.

O presente artigo apresenta uma metodologia de verificação de filtros digitais com implementação em ponto fixo, baseada em *Efficient SMT-Based Context-Bounded Model Checker* (ESBMC), que utiliza técnicas de *Bounded Model Checking* (BMC) e as teorias do módulo da satisfabilidade (*Satisfability Modulo Theories* - SMT) [2], [3]. Tal abordagem indica, de acordo com parâmetros de projeto de filtro previamente determinados, se a quantidade de bits utilizada na precisão finita é suficiente e não ocasionará erros ou comportamentos inesperados. A principal vantagem desta abordagem, sobre outras técnicas de análise de filtros [4], [5], é que as ferramentas de verificação formal podem fornecer informações precisas de como reproduzir um erro (por exemplo, valores das entradas do sistema).

Um verificador baseado na metodologia proposta foi implementado em ANSI-C e utilizado na verificação de diversos filtros digitais práticos, com o objetivo de validá-lo frente a projetos reais. Como resultado, tal verificador, em conjunto com as ferramentas de projeto tradicionais, proporcionou um esquema completo de sintetização de filtros, de acordo com as condições de aplicação.

Este artigo está organizado da seguinte forma. A seção II apresenta os esquemas de verificação disponíveis na literatura,

Mauro L. de Freitas⋆, Mikhail Y. R. Gadelha ⋆, Lucas Cordeiro⋆, Waldir S. da Silva Júnior⋆ e Eddie B. L. Filho⋆†. ⋆Universidade Federal do Amazonas - UFAM, Av. Gen. Rodrigo Octávio Jordão Ramos, 3000, Manaus - AM, 69077-000, Brasil. †Centro de Ciência, Tecnologia e Inovação do Pólo Industrial de Manaus, Rua Salvador, 391, Adrianópolis, Manaus-AM, 69057-040, Brasil. E-mails: maurokenny@gmail.com, mikhail@ufam.edu.br, lucascordeiro@ufam.edu.br, waldirsabino@gmail.com, eddie@ctpim.org.br.

destacando suas principais características. Na seção III, a técnica BMC é apresentada. Em seguida, na seção IV, o método proposto é descrito, e a seção V apresenta os resultados de simulações. Finalmente, as conclusões são expostas na seção VI.

II. Trabalhos relacionados

A aplicação de ferramentas que implementam a técnica BMC, para verificação de *software*, estão se tornando bastante populares, principalmente devido ao advento de solucionadores SMT sofisticados, que são construídos com base em solucionadores de satisfabilidade (SAT) [6] eficientes. Trabalhos anteriores, relacionados a BMC baseado em SMT, para software, tratam o problema de verificar programas ANSI-C que usam operações de bit, aritmética de ponto fixo e ponto flutuante, comparações e aritmética de ponteiros [2], [7]. No entanto, existe pouca evidência de trabalhos que abordam a verificação de propriedades (*e.g.*, de segurança e vivacidade) relacionadas à implementação de filtros digitais, em ANSI-C, principalmente quando se consideram palavras com comprimentos arbitrários.

Akbarpour e Tahar [8], [9] apresentam uma abordagem mecânica para a detecção de erros em projetos de filtros digitais, que é baseada em um provador de teoremas de lógica de mais alta ordem (*High Order Logic* - HOL). Os autores descrevem funções de avaliação que encontram os valores reais das saídas do filtro digital, através de representações em ponto fixo e ponto flutuante, com o intuito de definir um erro. Tal erro representa a diferença entre os valores encontrados, através desta função de avaliação, e a saída correspondente à especificação de projeto. Akbarpour e Tahar consideram a implementação de filtros digitais usando as três formas canônicas de realização: direta, paralela e cascata. No entanto, grande parte do processo de verificação é feito de forma manual, o que dificulta a aplicação dessa abordagem.

Recentemente, Cox, Sankaranarayanan e Chang [10] introduziram uma nova abordagem que utiliza análise de bit precisa para a verificação de implementações de filtros digitais, em ponto fixo. Essa abordagem é baseada na técnica de BMC e utiliza solucionadores SMT para checar condições de verificação, que são geradas a partir do projeto do filtro digital. Os autores mostram que tal abordagem é mais eficiente e produz menos alarmes falsos, quando comparada às que utilizam solucionadores de aritmética real. Entretanto, os trabalhos mencionados não abordam características intrínsecas dos filtros, como erros ou modificações relativas a pólo e zeros e resposta em frequência.

As constatações apresentadas nos parágrafos anteriores serviram de inspiração para o presente artigo, que tem como objetivo estender a abordagem proposta por Cox, Sankaranarayanan e Chang [10]. Em resumo, novas propriedades de filtros digitais a serem checadas, tais como as respostas em magnitude e fase, foram adicionadas, o que confere uma análise mais embasada na teoria de PDS e preenche as lacunas apresentadas pela literatura existente.

III. A Técnica BMC

No ESBMC, o programa que está sendo analisado é modelado por um sistema de transição de estados, que é gerado a partir do grafo de fluxo de controle do programa (*Control-Flow Graph* - CFG) [11]. O grafo de fluxo de controle do programa é gerado automaticamente, durante o processo de verificação. Um nó no CFG representa um atribuição (determinística ou não-determinística) ou uma expressão condicional, enquanto que uma aresta representa uma mudança no fluxo do programa.

Um sistema de transição de estados ***M = (S, T, S_0)*** é uma máquina abstrata, que consiste em um conjunto de estados *S*, onde $S_0 \subseteq S$ representa um conjunto de estados iniciais e $T \subseteq S \times S$ é a relação de transição. Um estado $s \in S$ consiste no valor do contador de programa *pc* e também nos valores de todas as variáveis do aplicativo. Um estado inicial s_0 atribui a localização inicial do programa no CFG. As transições são identificadas como $\gamma = (s_i, s_{i+1}) \in T$ entre dois estados s_i e s_{i+1}, com uma fórmula lógica $\gamma(s_i, s_{i+1})$ que contém as restrições dos valores do contador de programa e das variáveis do aplicativo.

Dada um sistema de transição *M*, uma propriedade ϕ e um limite *k*, o ESBMC desdobra o sistema *k* vezes e transforma o resultado em uma condição de verificação (*verification condition* - VC) ψ, de tal forma que ψ é satisfeita se, e somente se, ϕ possuir um contraexemplo de comprimento menor ou igual a *k* [2]. O problema da técnica de BMC é então formulado como

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

onde ϕ é uma propriedade, 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 passos j e $j+1$. Logo, $I(s_0) \wedge \bigwedge_{j=0}^{i-1} \gamma(s_j, s_{j+1})$ representa a execução de M, i vezes, e (1) só poderá ser satisfeita se, e somente se, para um $i \leq k$, exista um estado alcançável no passo em que ϕ é violada. Se (1) é satisfeita, então o ESBMC mostra um contra-exemplo, definindo quais os valores das variáveis necessárias para reproduzir o erro. O contraexemplo para uma propriedade ϕ é uma sequência de estados $s_0, s_1, \cdots, s_k$ com $s_0 \in S_0, e \gamma(s_i, s_{i+1})$ com $0 \leq i < k$. Se (1) não for satisfeita, pode-se concluir que nenhum estado com erro é alcançável em k passos ou menos.

IV. A nova Metodologia de Verificação

De maneira geral, a implementação em ponto fixo utiliza registradores padrões para armazenar as entradas e as saídas, ao longo dos somadores, multiplicadores e atrasos. Entretanto, os resultados desses elementos podem exceder os limites das variáveis alocadas, ou gerar valores diferentes do esperado, devido à precisão dos coeficientes ou ao número de bits associado. Como resultado, é possível que o resultado difira do especificado no projeto ou até mesmo que um filtro se torne instável, mesmo que o seu projeto não seja.

Tendo isso em foco, a metodologia de verificação proposta foi separada em três partes principais: verificação de magnitude e fase, estabilidade em pólos e zeros e verificação de *overflow*.

A. Verificação de magnitude e fase

Mudanças nos coeficientes, devido à quantização em ponto fixo, alteram a resposta em magnitude e fase [9]. Um exemplo disso pode ser visto na Fig. 1.

Nesta primeira abordagem, a entrada do sistema proposto é composta pelos coeficientes do filtro, em ponto flutuante, pelas propriedades de projeto, que devem ser analisadas de acordo com as condições adotadas, tais como banda de passagem, frequência de corte, banda de rejeição e os ganhos em cada região, e pela quantidade de bits utilizada na representação, em ponto fixo.

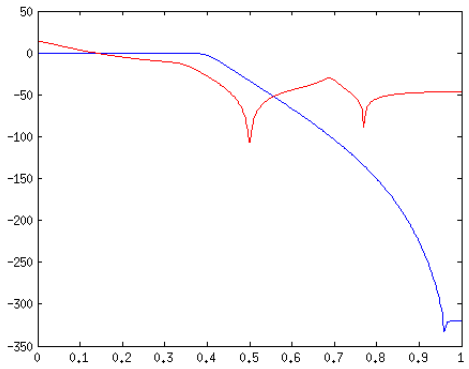

Fig. 1. Resposta em magnitude para um filtro IIR Chebyshev, de ordem 12. A curva em azul é a projetada e a em vermelho é o resultado obtido após a sua implementação em ponto-fixo, com um bit de sinal, sete para a parte inteira e seis para a fracionária.

Dado que N é o número de pontos a serem verificados na Transformada de Fourier de Tempo Discreto (Discrete-Time Fourier Transform - DTFT) [12], $h[n]$ é a resposta ao impulso do filtro e $H(k)$ o seu equivalente, no domínio da frequência, então

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

Além disso, supondo ω_p, ω_r e ω_c como as frequências digitais das bandas de passagem, de rejeiçao e de corte, respectivamente, e A_p, A_r e A_c como os seus ganhos a serem verificados, as expressões de assertivas que verificam as propriedades de magnitude e fase, de filtros passa-baixas e passa-altas, foram criadas como

$$\begin{aligned} I_{pb\_mag} \Leftrightarrow ((|H(k)| < A_p) \wedge (0 \leq \frac{2\pi k}{N} \leq \omega_p)) \\ \vee\ ((|H(k)| > A_c) \wedge (\frac{2\pi k}{N} = \omega_c)) \\ \vee\ ((|H(k)| > A_r) \wedge\ (\omega_r \leq \frac{2\pi k}{N} \leq \pi)), \end{aligned} \quad (3)$$

$$\begin{aligned} I_{pa\_mag} \Leftrightarrow ((|H(k)| > A_r) \wedge\ (0 \leq \frac{2\pi k}{N} \leq \omega_r)) \\ \vee((|H(k)| < A_c) \wedge (\frac{2\pi k}{N} = \omega_c)) \\ \vee((|H(k)| < A_p) \wedge\ (\omega_p \leq \frac{2\pi k}{N} \leq \pi)) \end{aligned} \quad (4)$$

e

$$I_{pb\_fase}, I_{pa\_fase} \Leftrightarrow |\sphericalangle H(k) - \sphericalangle H_{fixed}(k)| > limiar. \quad (5)$$

Caso as expressões sejam violadas, um erro é gerado, o que indica que a quantidade de bits é insuficiente para a representação dessas respostas, dadas as restrições iniciais de projeto.

B. Verificação de pólos e zeros

A estabilidade, com base nos pólos e zeros do sistema, também deve ser verificada. O processo envolve encontrar os pólos e o zeros do sistema, a partir da função de transferência dos filtros, e checar se o módulo de cada pólo é menor que 1. Para tal tarefa, um algoritmo de decomposição QR [13], da matriz companheira do polinômio formado pelos pólos, foi utilizado.

Dado um polinômio $p(t) = t^n + a_{n-1}t^{n-1} + \cdots + a_1 t + a_0$, a matriz companheira é definida como

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

Em seguida, a Matriz A é fatorada através da decomposição QR, de tal forma que $A = Q * R$, onde A é uma matrix de dimensões $N \times N$, Q é uma matriz quadrada ortogonal (onde $Q^T * Q = I$), de dimensões $N \times N$, e R é uma matriz triangular superior, também de dimensões $N \times N$. Em resumo, o Algoritimo é dado por [14]:

1) Sendo $A_m = Q_m * R_m$ a fatoração de A_m, criar $A_{m+1} = R_m * Q_m$;
2) Repetir o passo 1, até que os valores abaixo da diagonal principal de A_m sejam suficientemente pequenos ou não haja possibilidade de conversão dos valores. Os valores das raízes do polinômio estarão na diagonal principal da matriz A_m.

Na implementação realizada para este trabalho, utilizou-se um limite máximo de 60 iterações, pois essa precisão foi suficiente para a representação das raízes.

Tal processo é realizado para os polinômios formados pelos pólos e zeros do sistema, a fim de se retirar pólos e zeros com mesmo valor (estáveis). Por fim, há uma checagem para se verificar se cada pólo possui módulo menor que 1, o que determina o sistema como estável; caso contrário, este é instável [4].

C. Verificação de overflow

A terceira parte consiste na verificação de *overflow* após a quantização de coeficientes, o que, sem uma ferramenta computacional, é considerado uma tarefa árdua.

As operações de adição, subtração, multiplicação e divisão podem ser realizadas com uma representação em ponto fixo, de modo aproximado, para que esses valores permaneçam dentro dos intervalos impostos pela quantidade de bits. Caso essa condição seja violada, ocorre um *overflow*. Um dos erros, chamado de saturação, ocorre quando valores fora da faixa de bits são representados pelos limites máximos ou mínimos. É possível computar o limitante do módulo de entrada para a verificação de overflow, mas não se sabe qual entrada ocasionará este efeito. O *wrapping around* ocorre quando, ao se ultrapassar o limite, o valor máximo é atribuido a um valor mínimo e vice-versa [10]. A verificação possui, como entrada, números reais não determinísticos $\tilde{x}[n]$, além dos coeficientes $h[n]$ do filtro a ser verificado e o número de entradas N.

Todas as iterações realizadas para se obter a resposta da filtragem são verificadas, através da busca por violações de *overflow*, como em

$$I_{overflow} \Leftrightarrow ((\tilde{x}[n]h[0] > V_{max}) \vee (\tilde{x}[n-1]h[1] > V_{max}) \vee \ldots\ (\tilde{x}[n-N-1]h[N-1] > V_{max})) \quad (6)$$

e

$$I_{underflow} \Leftrightarrow ((\tilde{x}[n]h[0] < V_{min}) \vee (\tilde{x}[n-1]h[1] < V_{min}) \vee \ldots\ (\tilde{x}[n-N-1]h[N-1] < V_{min})). \quad (7)$$

Ao se detectar um erro, um contraexemplo é gerado. Tal contraexemplo consiste na lista de estados até a violação encontrada, o que permite saber quais foram as entradas que ocasionaram os erros, em uma ordem específica, e o valor de saída. Esta abordagem permite ao projetista saber, exatamente, em quais condições ocorreram erros de *overflow* e *underflow*, o que possibilita um melhor tratamento destas condições, a partir das novas informações obtidas.

V. Experimentos

Esta seção é composta por duas partes. A configuração de sistema está descrita na seção V-A, enquanto que a seção V-B descreve os resultados obtidos através da ferramenta ESBMC[1] [2], [3], com as modificações propostas neste artigo.

A. Configuração de sistema e preparação para os experimentos

O conjunto de testes de verificação de magnitude e fase foram separados em dois grupos principais: um de filtros IIR e outro de filtros FIR[2]. Cada conjunto, por sua vez, está dividido em duas categorias: passa-baixas e passa-altas, sendo três filtros de ordem pequena (ordem dois ou quatro) e três de ordem maior (doze ou trinta), em cada conjunto para diferentes frequências de corte.

Três tipos de filtros IIR foram utilizados: *Butterworth*, *Chebyshev* e Elíptico. Já para os filtros FIR projetados, os tipos utilizados foram *Equiripple*, Janela de *Hann* e *Maximally Flat*.

Ao todo foram criados 36 filtros estáveis durante a etapa de projeto, sendo 18 FIR e 18 IIR, com frequência de amostragem em 48 kHz. Todos os filtros foram gerados a partir da ferramenta MATLAB [1], utilizando-se a *toolbox* de Processamento de Sinais.

De modo a se explorar diferentes teorias utilizadas nos solucionadores SMT [2], números não inteiros foram codificados de duas formas diferentes: em binário (quando a aritmética de vetores de bits é utilizada) e também em real (ao se trabalhar com aritmética racional). A representação em ponto fixo foi realizada dividindo-se o número a ser representado em suas partes inteira I, com m bits, e fracionária F, com n bits [15]. Essa abordagem é representada através da tupla $\langle I, F \rangle$, que pode ser codificada tanto em vetores de bits quanto aritmética racional e é interpretada como $I + F/2^n$. Sendo assim, os valores representados devem estar entre os valores máximo e mínimo esperados, ou seja,

$$V_{max} = 2^m - 1/2^n, \quad (8)$$

$$V_{min} = -2^m \quad (9)$$

e

$$V_{min} \leq v_{fixed} \leq V_{max}. \quad (10)$$

Todos os experimentos foram conduzidos em um PC Intel Core i7-2600, de 3.40Ghz de *clock*, com 24GB de RAM e sistema operacional Ubuntu de 64-bits. Para todos os casos de teste, o tempo limite e o limite de memória foram ajustados para 1200 segundos e 24GB (22GB de RAM e 2GB de memória virtual), respectivamente[3].

B. Resultados obtidos

Esta subseção descreve os resultados obtidos ao se executarem os casos de testes com o ESBMC modificado, proposto neste trabalho, para a verificação dos filtros digitais.

A Tabela I resume os resultados da verificação de magnitude e fase, para filtros IIR. O é a ordem do filtro, FC é a Frequência de Corte, em Hertz, e TVM e SM são tempo de execução, em segundos, e o status para a magnitude, respectivamente. Como resultado do status, pode ocorrer Sucesso (S), erro na Faixa de Passagem (FP), erro na Faixa de Rejeição (FR) e erro na Frequência de Corte (FC). TVF e SF são o tempo de execução e o status para frequência, respectivamente, que possuem como resultado Sucesso (S) ou Falha (F). PF é a quantidade de bits utilizada na representação em ponto fixo. Os parâmetros de projeto para filtros passa-baixas, no que diz respeito às frequências de passagem, corte e rejeição digitais, foram fixados em $0,3$, $0,4$ e $0,5$ rad, respectivamente. O ganho mínimo, na banda de passagem, foi fixado em $-0,9$ dB; já as atenuações mínimas, nas bandas de corte e rejeição, foram fixadas em $-3,0$ e $-6,0$ dB, respectivamente. Com relação aos filtros passa-altas, parâmetros equivalentes foram adotados. A Tabela II sintetiza os resultados obtidos para filtros FIR.

A verificação de pólos e zeros ocorreu no conjunto de testes de filtros IIR e o resultado está resumido na Tabela III. TV é o tempo de verificação, em segundos, e SPZ é o status da verificação de pólos e zeros. Vale a pena destacar os resultados para filtros com frequências de corte em 100 Hz, pois foram encontrados erros de magnitude em todos os casos. Além disso, filtros IIR apresentaram maior quantidade de erros de fase, devidos principalmente às condições de projeto restritas e também à sua característica de não linearidade [16]. Todos os resultados foram verificados em comparativo com simulações no MATLAB de forma individual, para que os erros apontados pela ferramenta fossem checados.

A Tabela IV indica os resultados da verificação de *overflow*, na qual é possível visualizar a eficácia na detecção desse tipo de erro. Todos os filtros testados foram do tipo IIR, que também foram utilizados nos testes da Tabela I. O valor máximo de todos foi ajustado para $1,96$, com erros de *overflow* que variaram de ± 0.1 a ± 0.2.

[1] http://www.esbmc.org

[2] benchmarks podem ser acessados em http://www.esbmc.org

[3] esbmc --unwind *B* –timeout 20m –fixedbv –memlimit 24g

TABELA I

RESULTADOS DA VERIFICAÇÃO DE MAGNITUDE E FASE DOS FILTROS IIR.

Filtros IIR	O	FC	TVM	SM	TVF	SF	PF
lp2	2	9600	23.32	S	17.718	F	<1,5>
hp2	2	9600	23.22	S	43.983	F	<1,5>
lp2EST	2	100	21.34	FP	16.686	F	<1,5>
lp12	12	9600	152.62	S	252.93	F	<4,10>
hp12	12	9600	142.04	S	233.89	F	<4,10>
lp12EST	12	100	202.56	FP	225.29	F	<4,10>
lp4C	4	9600	101.55	FP	157.7	F	<4,10>
hp4C	4	9600	101.79	FP	46.27	F	<4,10>
lp4ESTC	4	100	95.95	FR	158.3	F	<4,10>
lp12C	12	9600	149.07	S	242.98	F	<4,10>
hp12C	12	9600	142.1	S	238.7	F	<4,10>
lp12ESTC	12	100	195.3	FP	237.33	F	<4,10>
lp4E	4	9600	94.06	S	152.65	F	<4,10>
hp4E	4	9600	96.95	FP	46.33	F	<4,10>
lp4ESTE	4	100	94.33	FP	157.04	F	<4,10>
lp12E	12	9600	206.18	FP	246.77	F	<4,10>
hp12E	12	9600	207.39	FP	233.47	F	<4,10>
lp12ESTE	12	100	193.22	FP	250.09	F	<4,10>

TABELA II

RESULTADOS DA VERIFICAÇÃO DE MAGNITUDE E FASE DOS FILTROS FIR.

Filtros FIR	O	FC	TVM	SM	TVF	SF	PF
fhp10	10	9600	96.15	S	196.54	S	<4,10>
fhp10Equi	10	9600	127.64	FR	200.34	S	<4,10>
fhp10Hann	10	9600	127.44	FP	193.41	S	<4,10>
fhp30	30	9600	318.3	S	72.96	S	<4,10>
fhp30Equi	30	9600	476.33	FP	73.43	S	<4,10>
fhp30Hann	30	9600	323.25	S	76.44	F	<4,10>
flp10	10	9600	96.79	S	195.62	S	<4,10>
flp10Equi	10	9600	126.07	FP	194.62	S	<4,10>
flp10EST	10	100	127.82	FP	192.29	S	<4,10>
flp10ESTEqui	10	100	126.59	FP	196.72	F	<4,10>
flp10ESTHann	10	100	128.78	FP	192.74	F	<4,10>
flp10Hann	10	9600	126.87	FP	190.81	F	<4,10>
flp30	30	9600	336.92	S	58.33	S	<4,10>
flp30Equi	30	9600	415.6	FP	72.41	S	<4,10>
flp30EST	30	100	468.91	FP	73.12	S	<4,10>
flp30ESTEqui	30	100	453.6	FP	75.31	F	<4,10>
flp30ESTHann	30	100	470.22	FP	75.74	F	<4,10>
flp30Hann	30	9600	323.25	S	75.77	F	<4,10>

TABELA III

RESULTADOS DA VERIFICAÇÃO DE MAGNITUDE E FASE DOS FILTROS IIR.

Filtro FIR	O	FC (Hz)	TV (s)	SPZ	PF
hp12	12	9600	0.14	F	<4,10>
hp12C	12	9600	0.14	S	<4,10>
hp12E	12	9600	0.14	F	<4,10>
hp2	2	9600	0.14	S	<1,5>
hp4E	4	9600	0.13	S	<4,10>
lp12	12	9600	0.14	F	<4,10>
lp12C	12	9600	0.14	F	<4,10>
lp12E	12	9600	0.14	F	<4,10>
lp12ESTC	12	100	0.14	F	<4,10>
lp12ESTE	12	100	0.14	F	<4,10>
lp2	2	9600	0.13	S	<1,5>
lp4E	4	9600	0.14	S	<4,10>

TABELA IV

RESULTADOS DA VERIFICAÇÃO DE OVERFLOW PARA FILTROS IIR.

Filtro	Ordem	Tempo (s)	PF	Status
lp2	2	372.03	<1,5>	S
hp2	2	361.34	<1,5>	F
lp4E	4	1135.596	<1,5>	F
hp4E	4	32.118	<1,5>	F

Nenhum caso de teste excedeu o tempo limite estipulado de 1200 segundos e não ocorreram casos com estouro de memória, durante os testes.

VI. CONCLUSÕES

O presente trabalho propôs uma metodologia para a verificação de parâmetros de projeto de filtros digitais, através da técnica BMC, que indica se a quantidade de bits utilizada na representação de coeficientes e amostras altera as características previamente especificadas. Durante as simulações, tanto filtros IIR quanto FIR foram abordados, de modo a se garantir o resultado de projetos reais, em diferentes realizações e aplicações. Os resultados mostram que é possível se detectar falhas em filtros de baixa e média ordem, com um tempo de verificação moderado. A principal contribuição deste trabalho reside na incorporação de parâmetros mais relacionados às teorias de projeto de filtros, como as respostas de magnitude e fase e a localização de pólos e zeros, o que complementa os testes de características mais computacionais, com *overflow*. Para trabalhos futuros, vislumbra-se a incorporação de outros parâmetros e a obtenção automática do número mínimo de bits para validação do projeto de filtros, em ponto fixo.

AGRADECIMENTOS

Os autores gostariam de agradecer ao CNPq e a CAPES, pelo auxílio financeiro.

REFERÊNCIAS

[1] Mathworks, Inc., "Matlab Getting Started Guide R2011b", USA, 2011.
[2] L. Cordeiro, B. Fischer e J. M. Silva, "SMT-Based Bounded Model Checking for Embedded ANSI-C Software", *In IEEE Transactions on Software Engineering (TSE)*, v. 38, pp. 957–974, IEEE, 2012. 1999.
[3] L. Cordeiro e B. Fischer, "Verifying Multi-threaded Software using SMT-based Context-Bounded Model Checking". *In Intl. Conf. on Software Engineering (ICSE)*, pp. 331–340, IEEE/ACM, 2011.
[4] A. Oppenheim, R. Schafer e J. Buck, "Discrete-time signal processing", *Prentice-Hall, Inc.*, Edição 2, 1999.
[5] W. T. Padgett and D. V. Anderson, "Fixed-point signal processing", Synthesis Lectures on Signal Processing, vol. 4, no. 1, pp. 1-133, 2009.
[6] L. de Moura e N. Bjørner, "Z3: An efficient SMT solver", *In TACAS*, LNCS 4963, pp. 337–340, 2008.
[7] E. Clarke, D. Kroening, e F. Lerda, "A tool for checking ANSI-C programs", *In TACAS*, LNCS 2988, pp. 168–176, 2004.
[8] B. Akbarpour e S. Tahar: "Error Analysis of Digital Filters Using Theorem Proving". *In TPHOLs*, pp. 1–17, 2004.
[9] B. Akbarpour e S. Tahar. "Error analysis of digital filters using HOL theorem proving". *In J. Applied Logic* 5(4): pp. 651–666, 2007.
[10] A. Cox, S. Sankaranarayanan, e Bor-Yuh E. Chang, "A bit too precise? Bounded verification of quantized digital filters", *In TACAS*, LNCS 7214, pp. 33–47, 2012.
[11] S. S. Muchnick, "Advanced compiler design and implementation.", *Morgan Kaufmann Publishers Inc.*, 1997.
[12] P. Diniz, E. Silva e S. Netto, "Processamento Digital de Sinais - Projeto e Análise de Sistemas", *Bookman Editora*, 2002.
[13] S. Chandrasekaran, M. Gu, J. Xia e J. Zhu. "A Fast QR Algorithm for Companion Matrices. Operator Theory: Advances and Applications", vol 179, pp. 111-143.
[14] D. C. Lay. Linear Algebra: Case Studies and Applications.
[15] D. Kroening e O. Strichman, Decision Procedures: An Algorithmic Point of View. Springer, 2008.
[16] M. Hraschan e M. Viertler, "FIR vs IIR Filter", http://secure.spsc.tugraz.at/courses/vwa/papergr6, (última visita 25/04/2013)