Uso de Estratégias de Multiplicação por Múltiplas Constantes para Implementação de Filtros Volterra

Rogério Paludo e Eduardo Luiz Ortiz Batista

***Resumo*— Este trabalho é dedicado ao desenvolvimento de uma nova estratégia eficiente para implementação de filtros Volterra em plataformas de hardware flexíveis como os FPGAs (*field-programmable gate arrays*). Essa estratégia é baseada no uso de técnicas de multiplicação por múltiplas constantes (MCM), nas quais deslocamentos, somas e reuso de hardware são utilizados para realizar multiplicações por constantes. Uma implementação transposta para *kernels* Volterra também é desenvolvida no presente trabalho, visando potencializar a aplicação de técnicas de MCM. Resultados experimentais são apresentados visando demonstrar a efetividade da estratégia proposta.**

***Palavras-Chave*— Filtro Volterra, FPGA, MCM.**

***Abstract*— This work is focused on the development of a new effective strategy to implement Volterra filters using flexible or configurable hardware platforms such as field-programmable gate arrays (FPGAs). The proposed strategy is based on multiple-constant-multiplication (MCM) techniques, which utilize shifts, sums, and hardware reuse for effectively carrying out multiplications by constants. A transposed implementation for Volterra kernels is also introduced in this work aiming to further enhance the results obtained using MCM techniques. Experimental results demonstrate the effectiveness of the proposed strategy.**

***Keywords*— Volterra filter, FPGA, MCM.**

I. Introdução

Diversos problemas de processamento de sinais requerem soluções que vão além das tradicionais soluções lineares. A dificuldade inicial para resolver tais problemas é a seleção de uma dentre as diferentes classes de soluções ou filtros não lineares [1]. Nesse contexto, uma das classes mais conhecidas é a dos filtros Volterra, os quais têm a capacidade de serem aproximadores universais para sistemas discretos, causais e com memória finita [2]. Tal capacidade está associada a um alto custo computacional, em função do elevado número de parâmetros (coeficientes) [1]. Como consequência, um significativo esforço de pesquisa tem sido realizado nas últimas décadas visando o desenvolvimento de implementações de complexidade reduzida para filtros Volterra [3]–[10].

Dentre as abordagens comumente utilizadas para implementação de filtros Volterra com complexidade reduzida, podemos citar as abordagens que exploram algum tipo de esparsidade do *kernel* (veja [3] e referências) ou mesmo abordagens baseadas em decomposições matriciais ou tensoriais [4]–[8]. Abordagens que exploram plataformas de hardware flexíveis, como os *field-programmable gate arrays* (FPGAs), também têm sido exploradas [9], [10]. Essas plataformas permitem a adoção de estratégias mais heterodoxas para implementação de filtros Volterra, como por exemplo o uso de tamanhos de palavra distintos para diferentes partes do filtro, conforme proposto em [10]. Assim, torna-se possível atingir reduções de complexidade que vão além dos limites impostos pelas plataformas tradicionais (DSPs e outros processadores).

Rogério Paludo e Eduardo L. O. Batista estão vinculados ao LINSE-Laboratório de Circuitos e Processamento de Sinais do Departamento de Engenharia Elétrica e Eletrônica da Universidade Federal de Santa Catarina, Florianópolis, SC, Brasil (e-mails: rogerio.pld@gmail.com e eduardo.batista@ufsc.br).

O presente trabalho de pesquisa é focado no desenvolvimento de uma nova abordagem eficiente para implementação de filtros Volterra em plataformas de hardware flexíveis como os FPGAs ou mesmo em *application-specific integrated circuits* (ASICs). A abordagem proposta envolve o uso de estratégias de multiplicação por múltiplas constantes (*multiple constant multiplication* - MCM) [11], as quais são baseadas na realização de multiplicações por constantes usando deslocamentos e somas, além da eliminação de hardware redundante via reuso. Esse tipo de estratégia é bastante utilizado, por exemplo, para implementação de filtros FIR de forma eficiente [11]. Visando potencializar o uso de estratégias de MCM no caso dos filtros Volterra, uma nova forma de implementação de *kernels* Volterra, denominada implementação transposta, é introduzida no presente trabalho. Como resultado, torna-se possível a obtenção de implementações bastante eficientes com respeito a complexidade computacional, precisão numérica e velocidade de operação, esta última medida em frequência máxima de *clock* ou atraso crítico. Por simplicidade, a abordagem proposta será desenvolvida considerando *kernels* de segunda ordem (quadráticos) de filtros Volterra, sendo a extensão para *kernels* de maiores ordens discutida brevemente.

O restante deste trabalho está estruturado conforme descrito a seguir. A Seção II é dedicada à descrição geral da fundamentação matemática necessária para o desenvolvimento do trabalho. Na Seção III, as principais contribuições são apresentadas, sendo elas a implementação transposta de *kernels* Volterra e a estratégia geral para implementação baseada em MCM. Na Seção IV, os resultados experimentais obtidos são apresentados, enquanto a Seção V é dedicada à apresentação das conclusões finais do trabalho.

II. Fundamentação Teórica

Esta seção apresenta a base teórica necessária para o desenvolvimento do presente trabalho. Assim, primeiramente o filtro Volterra é descrito com destaque para o *kernel* quadrático, seguido por uma discussão acerca da implementação de tal *kernel* a partir de filtros FIR. Também nesta seção, uma visão geral das técnicas de MCM é apresentada em conjunto com a descrição do seu uso para implementação de filtros FIR.

A. Filtros Volterra

De maneira geral, a saída $y(n)$ de um filtro Volterra de ordem P é dada pela soma das saídas de *kernels* polinomiais com ordens 1 a P. Em sua forma triangular [1], ou com redundância removida [12], tem-se a seguinte relação de entrada e saída:

$$y(n) = \sum_{p=1}^{P} y_p(n) \tag{1}$$

com

$$y_p(n) = \sum_{m_1=0}^{N} \cdots \sum_{m_p=m_{p-1}}^{N} h_p(m_1, \ldots, m_p) \prod_{k=1}^{p} x(n-m_k) \quad (2)$$

representando a saída do *kernel* de ordem p, $h_p(m_1, \ldots, m_p)$ os coeficientes de tal *kernel*, e N o tamanho de memória do filtro. O *kernel* de primeira ordem corresponde a um filtro FIR, sendo portanto linear. Os demais *kernels* (com $p \geq 2$) são não lineares, envolvendo produtos de amostras do sinal de entrada.

Como mencionado anteriormente, por simplicidade, o presente trabalho é focado na implementação de *kernels* de segunda ordem ou quadráticos, para os quais a relação de entrada e saída é dada por

$$y_2(n) = \sum_{m_1=0}^{N} \sum_{m_2=m_1}^{N} h_2(m_1, m_2) x(n-m_1) x(n-m_2). \quad (3)$$

Considerando m_1 e m_2 como coordenadas de linha e coluna, é possível definir a seguinte matriz de coeficientes:

$$\mathbf{H}_2 = \begin{bmatrix} h_2(0,0) & h_2(0,1) & \cdots & h_2(0,N-1) \\ 0 & h_2(1,1) & \cdots & h_2(1,N-1) \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & h_2(N-1,N-1) \end{bmatrix}. \quad (4)$$

B. Implementação do Kernel Quadrático Usando Filtros FIR

É comum o uso de estruturas baseadas em filtros FIR para implementação de *kernels* Volterra não lineares, em função da vasta literatura disponível com relação à realização prática de tais filtros. Em geral, são duas as principais abordagens diretas para realização desse tipo de implementação: uma baseada no uso de coordenadas diagonais [13] e outra em fatoração de *kernels* [8]. Abordagens baseadas em decomposições matriciais ou tensoriais [5]–[7], [14] também podem ser usadas com tal finalidade. No entanto, essas abordagens estão fora do escopo do presente trabalho por envolverem também aproximações ou simplificações na implementação dos *kernels*.

Para se obter a implementação baseada em coordenadas diagonais, conforme descrito em [1], [13], a relação de entrada e saída do *kernel* quadrático, dada por (3), é reescrita da seguinte forma a partir de um processo de substituição de variáveis:

$$y_2(n) = \sum_{r=0}^{N-1} \mathbf{h}'^{\mathrm{T}}_{2,r} \mathbf{x}'_{2,r}(n) \quad (5)$$

com $\mathbf{h}'_{2,r} = [h_2(0,r)\ h_2(1,r+1)\ \cdots\ h_2(N-1-r,N-1)]^{\mathrm{T}}$ representando um vetor parcial de coeficientes e

$$\mathbf{x}'_{2,r}(n) = [x(n)x(n-r)\ \ x(n-1)x(n-r-1)\ \cdots \\ x(n-N+1+r)x(n-N+1)]^{\mathrm{T}} \quad (6)$$

um vetor de entrada correspondente. Como (6) é composto por valores atrasados de $x(n)x(n-r)$, observa-se que o r-ésimo elemento do somatório em (5) corresponde a um filtro FIR com sinal de entrada $x(n)x(n-r)$ e vetor de coeficientes $\mathbf{h}'_{2,r}$. Assim tem-se, a partir de (5), a implementação do *kernel* Volterra quadrático na forma ilustrada na Fig. 1. Essa implementação é dita baseada em coordenadas diagonais uma vez que $\mathbf{h}'_{2,r}$ corresponde à r-ésima diagonal da matriz de coeficientes $\mathbf{H}_2$ ($r = 0$ corresponde à diagonal principal).

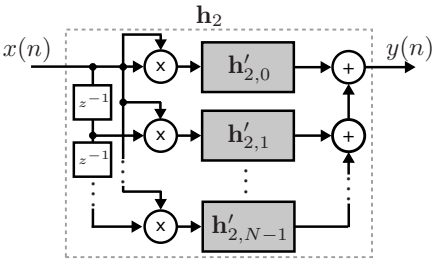

Fig. 1. Implementação do *kernel* quadrático baseada em coordenadas diagonais.

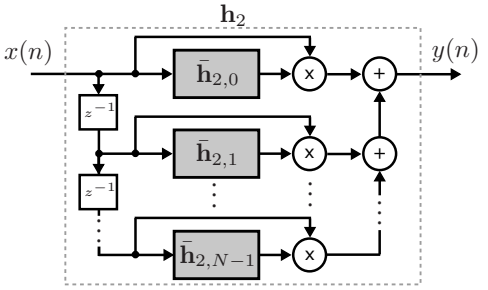

Fig. 2. Implementação do *kernel* quadrático baseada em fatoração de *kernels*.

Uma segunda forma de se implementar *kernels* Volterra usando filtros FIR paralelos é obtida a partir do procedimento, descrito em [8], de fatoração de um *kernel* de ordem p em um conjunto de *kernels* de ordem $p-1$. Aplicando tal fatoração ao *kernel* quadrático, obtém-se a seguinte relação de entrada e saída:

$$y_2(n) = \sum_{s=0}^{N-1} x(n-s) \bar{\mathbf{h}}^{\mathrm{T}}_{2,s} \bar{\mathbf{x}}_{2,s}(n) \quad (7)$$

com

$$\bar{\mathbf{h}}_{2,s} = [h_2(s,s)\ h_2(s,s-1)\ \cdots\ h_2(s,N-1)]^{\mathrm{T}} \quad (8)$$

e $\bar{\mathbf{x}}_{2,s}(n) = [x(n-s)\ x(n-s-1) \cdots x(n-N+1)]^{\mathrm{T}}$. A caraterística de linha de atraso de $\bar{\mathbf{x}}_{2,s}(n)$ deixa evidente que o produto $\bar{\mathbf{h}}^{\mathrm{T}}_{2,s} \bar{\mathbf{x}}_{2,s}(n)$ corresponde a um filtro FIR. Assim, (7) leva à estrutura para implementação do *kernel* quadrático apresentada na Fig. 2.

C. Implementação de Filtros FIR usando Técnicas de MCM

As técnicas de multiplicação por múltiplas constantes (*multiple constant multiplication* - MCM) são baseadas no reuso de hardware em situações onde uma única variável é multiplicada por um grande número de valores constantes [11]. Nessas situações, além da possibilidade de fazer multiplicações usando deslocamentos e somas, é possível também fazer o reuso de deslocamentos e/ou somas que são comuns às diferentes multiplicações. Como resultado, obtém-se reduções de custo computacional, via reduções de complexidade e área de circuito, bem como ganhos de desempenho, em função de redução da profundidade (atraso crítico) do circuito. Presumivelmente, as técnicas de MCM são voltadas às plataformas de implementação em hardware que permitem tal tipo de reuso de forma eficiente, como é o caso dos FPGAs e ASICs.

Um importante nicho de aplicação de técnicas de MCM é para implementação de filtros FIR [11]. Para tal, utiliza-se a implementação FIR transposta a qual está ilustrada na Fig. 3

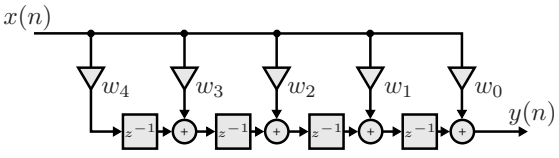

Fig. 3. Implementação de um filtro FIR na forma transposta.

para um filtro com tamanho de memória $N = 5$. Nota-se, a partir dessa figura, uma situação típica para uso eficiente de técnicas de MCM, uma vez que tem-se a multiplicação de uma variável [$x(n)$, a amostra atual do sinal de entrada] por múltiplas constantes ($w_0, w_1, \ldots, w_4$, os coeficientes do filtro). A implementação direta de filtros FIR [15] não é interessante para o uso de técnicas de MCM, uma vez que nela cada coeficiente é multiplicado por uma amostra atrasada do sinal de entrada [i.e., w_0 por $x(n)$, w_1 por $x(n-1)$, etc].

No presente trabalho, duas abordagens de MCM serão consideradas: Spiral [11] e RPAG [16]. Essas abordagens são do tipo *adder graph approaches* (AGAes) e assim são baseadas na busca por grafos de somas mínimos levando em conta restrições de profundidade e/ou número de somas. O uso dessas abordagens é justificado pelo fato que os trabalhos mais recentes da literatura têm demonstrado uma superioridade das AGAes em comparação com as suas concorrentes principais, que são as abordagens baseadas na eliminação de subexpressões comuns [17].

III. Implementação Eficiente de Filtros Volterra Usando MCM e Kernels Transpostos

Esta seção é dedicada ao desenvolvimento de uma nova abordagem para implementação eficiente de filtros Volterra usando técnicas de MCM. Esse tipo de implementação pode ser realizado simplesmente a partir da aplicação de técnicas de MCM em cada um dos filtros FIR que compõem implementações como as ilustradas nas Figs. 1 e 2. No entanto, nessas implementações, cada filtro FIR é submetido a um sinal de entrada diferente e, portanto, as abordagens de MCM precisam ser aplicadas de forma independente a cada filtro, limitando o reuso de hardware. Visando então potencializar a aplicação de técnicas de MCM em *kernels* Volterra, uma implementação análoga à FIR transposta é introduzida nesta seção, inicialmente com foco no *kernel* quadrático. Em seguida, uma breve discussão acerca da generalização para outros *kernels* não lineares é apresentada. Finalmente, a abordagem proposta é descrita de forma resumida.

A. Kernels Volterra Quadráticos Transpostos

Para o desenvolvimento de uma implementação transposta para o *kernel* quadrático, considera-se inicialmente a relação de entrada e saída de tal *kernel* escrita como (7). Então, reescrevendo o produto interno $\bar{\mathbf{h}}_{2,s}^{\mathrm{T}}\bar{\mathbf{x}}_{2,s}(n)$ na forma de somatório e movendo $x(n-s)$ para dentro desse somatório, obtém-se

$$y_2(n) = \sum_{s=0}^{N-1}\sum_{k=0}^{N-1-s} h_{2(s,s+k)}x(n-s)x(n-s-k). \quad (9)$$

Utilizando o operador de atraso (*lag* ou *backshift*), o qual é definido como $L^k x(n) = x(n-k)$ [18], é possível reescrever (9) como

$$y_2(n) = \sum_{s=0}^{N-1}\sum_{k=0}^{N-1-s} h_{2(s,s+k)}L^s[x(n)x(n-k)]. \quad (10)$$

Considerando que ambos L^s e $x(n)$ não dependem de k, eles podem ser movidos para fora do somatório interno de (10), resultando em

$$y_2(n) = \sum_{s=0}^{N-1} L^s\left[x(n)\sum_{k=0}^{N-1-s} h_{2(s,s+k)}x(n-k)\right]. \quad (11)$$

Note agora que o somatório mais interno em (11) corresponde a um filtro FIR com vetor de coeficientes dado por (8) e vetor de entrada dado por

$$\hat{\mathbf{x}}_{2,s}(n) = [x(n)\; x(n-1)\cdots x(n-N+s+1)]^{\mathrm{T}}. \quad (12)$$

Denotando a saída de tal filtro por $\hat{y}_{2,s}$, i.e. $\hat{y}_{2,s} = \bar{\mathbf{h}}_{2,s}^{\mathrm{T}}\hat{\mathbf{x}}_{2,s}(n)$, torna-se possível reescrever (11) como

$$y_2(n) = \sum_{s=0}^{N-1} L^s[x(n)\hat{y}_{2,s}(n)]. \quad (13)$$

Expandindo o somatório em (13) e isolando L de forma sucessiva, obtém-se

$$\begin{aligned} y_2(n) = {} & x(n)\hat{y}_{2,0}(n) + L[x(n)\hat{y}_{2,1}(n) + L[x(n)\hat{y}_{2,2}(n) \\ & + L[x(n)\hat{y}_{2,3}(n) + \cdots + L[x(n)\hat{y}_{2,N-1}(n)]\cdots]]]. \end{aligned} \quad (14)$$

A partir de (14), nota-se que um *kernel* Volterra quadrático pode ser implementado conforme ilustrado na Fig. 4.

B. Kernels Volterra Transpostos com Ordens Superiores

Note que o ponto de partida para o desenvolvimento da implementação transposta do *kernel* quadrático é a representação da sua relação de entrada e saída na forma de (7), a qual é obtida a partir do procedimento de fatoração de *kernels* descrito em [8]. Assim, a extensão da implementação transposta para *kernels* de maior ordem pode ser feita também a partir das suas implementações fatoradas descritas em [8]. Para o *kernel* cúbico (de terceira ordem), por exemplo, partindo da sua representação fatorada em *kernels* quadráticos e realizando um procedimento similar ao descrito na seção anterior, é possível obter uma estrutura de implementação transposta. Essa estrutura é similar à da Fig. 4, porém com os blocos destacados em cinza correspondendo a *kernels* quadráticos ao invés de filtros FIR. Então, usando a própria estrutura da Fig. 4 para implementação de tais *kernels* quadráticos, obtém-se uma implementação transposta para o *kernel* cúbico composta exclusivamente por filtros FIR.

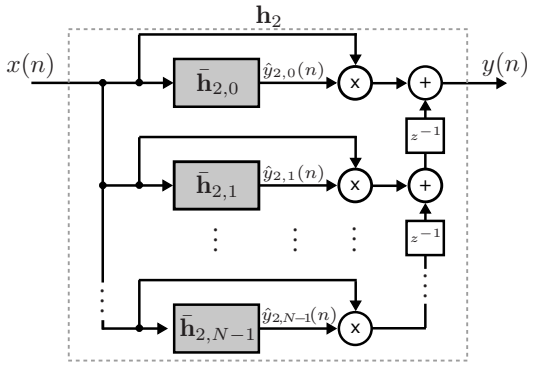

Fig. 4. Implementação de um *kernel* Volterra quadrático na forma transposta.

C. Resumo da Abordagem Proposta

De maneira geral, a abordagem para implementação eficiente de filtros Volterra proposta neste trabalho envolve o uso da implementação transposta de *kernels* Volterra, introduzida nas seções anteriores, em conjunto com abordagens de MCM para redução de custo computacional. Em tal implementação, absolutamente todos os coeficientes do *kernel* são multiplicados por $x(n)$, potencializando o reuso de hardware via técnicas de MCM. Assim, torna-se possível obter altas taxas de reuso de hardware, resultando em implementações muito eficientes de *kernels* Volterra voltadas a plataformas de hardware flexíveis como os FPGAs. É importante ressaltar ainda que o uso da implementação transposta para *kernels* Volterra não implica custos adicionais em termos de coeficientes, elementos de atraso ou operações aritméticas, uma vez que as estruturas das Figs. 1, 2 e 4 são similares nesses aspectos.

IV. Resultados Experimentais

As análises experimentais realizadas neste trabalho levam em consideração um modelo agnóstico de tecnologia baseado em número total de somas completas (*full-adders*). Tal modelo está relacionado diretamente aos custo e atraso crítico observados para as arquiteturas consideradas quando estas são mapeadas diretamente em FPGAs ou ASICs. Nesse contexto, primeiramente, é avaliada a aplicação dos métodos MCM (RPAG e Spiral) às implementações apresentadas nas Figs. 1 (denotada como diagonal), 2 (fatorada) e 4 (transposta). Essa avaliação considera, como referência, a implementação convencional baseada em representações canônicas com sinal (CSD) [19]. Em tal implementação, agrupamentos de 1's são recodificados para reduzir complexidade na implementação usando deslocamentos e somas [e.g., $(15)_{10} = (1111)_2 = 16 - 1 = (\bar{1}001)_{\text{CSD}}$]. Posteriormente, a realização completa de um *kernel* Volterra e seu custo equivalente são avaliados. Finalmente, o impacto de realizações em ponto fixo na precisão dos filtros é avaliado experimentalmente em comparação com os resultados obtidos com precisão dupla (*double*) em ponto flutuante. Todos os resultados consideram um *kernel* Volterra quadrático com $N = 25$ (total de 325 coeficientes), obtido a partir da modelagem adaptativa de um adaptador para telefonia analógica (ATA) de um sistema VoIP. Tal *kernel* é normalizado pela norma Frobenius das sua matriz de coeficientes $\|\mathbf{H}_2\|_F$ e quantizado em faixas dinâmicas de 8, 12 e 16 bits.

A. Aplicação de Métodos MCM e Implementação Transposta

A Fig. 5 mostra os resultados de total de somas completas e profundidade obtidos com aplicação de técnicas de MCM às estruturas aqui consideradas (diagonal, fatorada e transposta) em comparação com a implementação convencional (CSD). A partir dessa figura, observa-se as significativas reduções tanto de complexidade (97%, 89% e 82% para 8, 12 e 16 bits, respectivamente) quanto de profundidade (33%, 40% e 57%) obtidas usando as técnicas de MCM. Além disso, nota-se que a implementação transposta, proposta neste trabalho, é a que permite a obtenção de realizações com menor custo, a partir do favorecimento do compartilhamento de hardware. Comparando as técnicas de MCM utilizadas, observa-se que a RPAG é consistentemente melhor em obter menor profundidade de somas, enquanto a Spiral atinge menor custo computacional.

B. Avaliação de Custo Total

O objetivo agora é fazer uma avaliação do impacto da aplicação de métodos de MCM no custo total de implementação do *kernel* quadrático aqui considerado. Para tal, considera-se um cenário onde, além do circuito para multiplicação por constantes, a implementação CSD e as baseadas em MCM usam ainda: 1 multiplicador de W bits (para fazer as 25 multiplicações entre variáveis), 13 somadores de $2W$ bits (para as 325 somas dos resultados das multiplicações por constantes) e 1 somador de W bits (para as somas das saídas de cada ramo), com W representando o tamanho de palavra utilizado. Com essa escolha, tem-se uma implementação sequencial e a necessidade de 25 ciclos de *clock* para o cálculo da saída do filtro. Uma implementação direta (sem uso de multiplicações por constante usando deslocamentos) também é considerada. Nesse caso, são usados 14 multiplicadores de W bits, 13 somadores de $2W$ bits e 1 somador de W bits, visando obter o mesmo $throughput$ das demais implementações (i.e., cálculo da saída em 25 ciclos de *clock*). O custo considerado para um multiplicador é de W^2 somadores completos, enquanto que para cada somador tem-se um custo de W somadores completos. Os custos dos blocos de controle necessários para implementação sequencial dos circuitos não são considerados uma vez que eles tendem a ser parecidos para as diferentes implementações. Os resultados obtidos estão apresentados na Fig. 6, de onde observa-se que a estratégia proposta (Transposta Spiral) é a que leva aos melhores resultados (reduções entre 68% e 74% em comparação com a implementação direta e entre 55% e 63% em relação à CSD).

C. Precisão em Realizações de Ponto Fixo

Visando analisar a precisão das implementações de ponto fixo obtidas, considera-se um modelo numérico equivalente em que as operações são inteiras com truncamento, saturação e *clipping*. Nas operações de multiplicação por constante, considera-se o dobro da faixa dinâmica na saída com truncamento para valores inteiros em $[-1, 1[$. Escalonamentos com saturação, para evitar oscilações por variação de sinal em representações complemento a dois, são realizados após produtos internos e multiplicações. Por fim, na saída de cada ramo (filtro FIR) que compõe o *kernel* quadrático, é aplicado *clipping*, evitando oscilações de representação. Nessas condições, o erro médio quadrático normalizado é calculado por $\text{NMSE} = \sum({y_2}^2 - \hat{y}_2^2)/\sum {y_2}^2$, em que y_2 representa os valores de saída do *kernel* implementado com precisão dupla e $\hat{y}_2$ os valores da implementação com ponto fixo. Além disso, são registradas as porcentagens de operações numéricas que saturam para os limites de representação.

Os resultados para um sinal de entrada com 10000 amostras normalmente distribuídas com média 0 e variância $\sigma^2 = 0.5$ são apresentados na Fig. 7. A partir dessa figura, nota-se que a implementação baseada em cordenadas diagonais (diagonal) é a que leva aos piores resultados em termos de NMSE, enquanto que melhores resultados são obtidos usando tanto a implementação fatorada quanto a transposta proposta neste trabalho. Além disso, a probabilidade de saturação observada é também menor para as implementações fatorada e transposta.

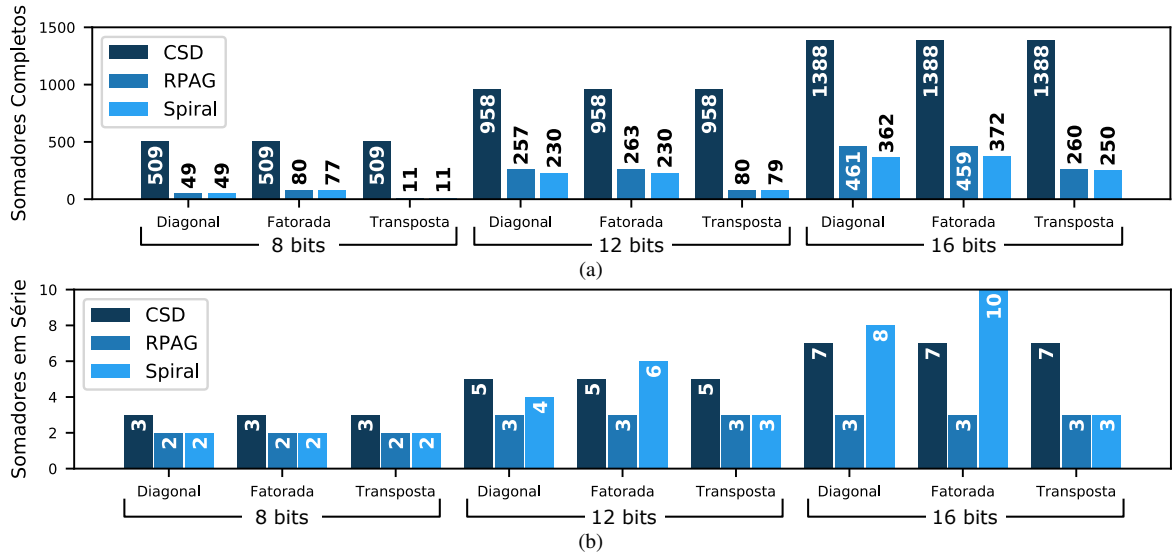

Fig. 5. Resultados experimentais para (a) total de somadores completos e (b) profundidade de somas (somadores em série).

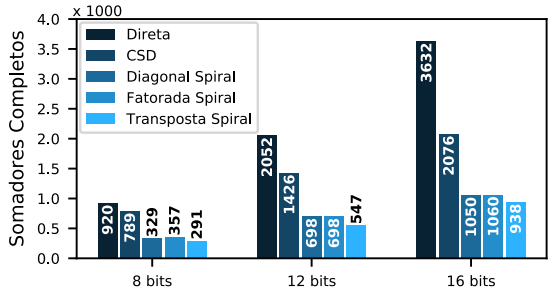

Fig. 6. Custo total de implementação para as estruturas consideradas.

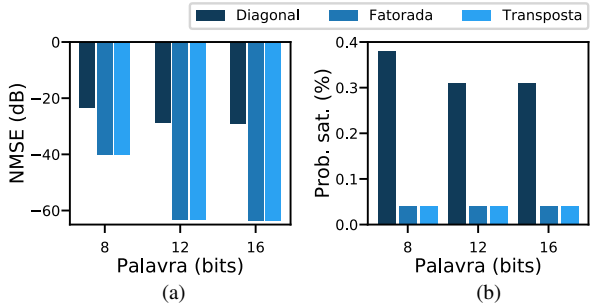

Fig. 7. Precisão e probabilidade de saturação para as diferentes implementações usando MCM.

V. Conclusões

O presente trabalho de pesquisa foi dedicado ao desenvolvimento de uma estratégia eficiente para implementação de filtros Volterra baseada no uso de técnicas de MCM. Para tal, uma nova forma de implementação de *kernels* Volterra foi desenvolvida, a qual é análoga à implementação transposta de filtros FIR no sentido que todos os coeficientes do *kernel* são multiplicados pela mesma amostra do sinal de entrada. Tal implementação potencializa a aplicação das técnicas de MCM, levando a estruturas de implementação mais eficientes. Resultados experimentais foram apresentados, mostrando a eficácia da abordagem proposta tanto em termos de custo computacional quanto de precisão.

Referências

[1] V. J. Mathews and G. L. Sicuranza, *Polynomial Signal Processing*. New York: John Wiley & Sons, Inc., 2000.

[2] A. Carini and G. L. Sicuranza, "Fourier nonlinear filters," *Signal Processing*, vol. 94, pp. 183–194, Jan. 2014.

[3] E. L. O. Batista and R. Seara, "On the performance of adaptive pruned Volterra filters," *Signal Processing*, vol. 93, no. 7, pp. 1909–1920, Jul. 2013.

[4] H.-H. Chiang, C. L. Nikias, and A. N. Venetsanopoulos, "Efficient implementations of quadratic digital filters," *Trans. Acoust., Speech, Signal Process.*, vol. 34, no. 6, pp. 1511–1528, Dec. 1986.

[5] S. Marsi and G. L. Sicuranza, "On reduced-complexity approximations of quadratic filters," in *Conf. Rec. 27th Asilomar Conf. Signals, Syst. Comput.*, vol. 2, Pacific Grove, CA, Nov. 1993, pp. 1026–1030.

[6] T. M. Panicker, V. J. Mathews, and G. L. Sicuranza, "Adaptive parallel-cascade truncated Volterra filters," *IEEE Trans. Signal Process.*, vol. 46, no. 10, pp. 2664–2673, 1998.

[7] G. Favier, A. Y. Kibangou, and T. Bouilloc, "Nonlinear system modeling and identification using Volterra-PARAFAC models," *Int. J. Adapt. Control Signal Process.*, vol. 26, no. 1, pp. 30–53, Jan. 2012.

[8] E. L. O. Batista and R. Seara, "A reduced-rank approach for implementing higher-order Volterra filters," *EURASIP J. Adv. Signal Process.*, no. 118 (2016), pp. 1–8, Nov. 2016.

[9] Y. Lou, C. L. Nikias, and A. N. Venetsanopoulos, "Efficient VLSI array processing structures for adaptive quadratic digital filters," *Circuits, Syst. Signal Process.*, vol. 7, no. 2, Jun. 1988.

[10] F. D. R. Oliveira, E. L. O. Batista, and R. Seara, "Effective hardware implementation of volterra filters based on reduced-rank approaches," *Electronics Letters*, vol. 54, no. 16, pp. 1005–1007, 2018.

[11] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplication," *ACM Trans. Algorithms*, vol. 3, no. 2, p. 11–es, May 2007.

[12] E. L. O. Batista, O. J. Tobias, and R. Seara, "A sparse-interpolated scheme for implementing adaptive Volterra filters," *IEEE Trans. Signal Process.*, vol. 58, no. 4, pp. 2022–2035, Apr. 2010.

[13] G. M. Raz and B. D. Van Veen, "Baseband Volterra filters for implementing carrier based nonlinearities," *IEEE Trans. Signal Process.*, vol. 46, no. 1, pp. 103–114, Jan. 1998.

[14] E. L. O. Batista and R. Seara, "A novel reduced-rank approach for implementing volterra filters," in *2016 24th European Signal Processing Conference (EUSIPCO)*, 2016, pp. 1778–1782.

[15] P. S. R. Diniz, E. A. B. da Silva, and S. L. Netto, *Digital Signal Processing*, 2nd ed. Cambridge University Press, 2010.

[16] M. Kumm, *Multiple Constant Multiplication Optimizations for Field Programmable Gate Arrays*, 1st ed. Wiesbaden, Germany: Springer Vieweg, 2016.

[17] P. K. Meher, C.-H. Chang, O. Gustafsson, A. Vinod, and M. Faust, *Shift-Add Circuits for Constant Multiplications*. John Wiley & Sons, Ltd, 2017, ch. 2, pp. 33–76.

[18] J. D. Hamilton, *Time Series Analysis*, 1st ed. Princeton, NJ: Princeton University Press, 1994.

[19] R. Hartley and K. K. Parhi, *Canonic Signed Digit Arithmetic*. Boston, MA: Springer US, 1995, pp. 217–252.