Denoising de Áudio com DWT/MODWT e Implementação em FPGA

Matheus Araújo de Oliveira, Walter Antônio Gontijo, Sidnei Noceti Filho e Eduardo L. O. Batista

***Resumo*— Este artigo apresenta a implementação em FPGA de um algoritmo de redução de ruído em sinais de áudio, utilizando variações da Transformada *Wavelet* Discreta (DWT). Inicialmente, o algoritmo de *denoising* considerando a DWT/MODWT é simulado em MATLAB para obter os diferentes parâmetros que resultem no melhor desempenho. Na sequência, o algoritmo resultante é sintetizado em FPGA, bem como é avaliado seu comportamento em simulação. Finalmente, seu desempenho é avaliado em um kit de desenvolvimento. Os resultados experimentais obtidos demonstram o bom funcionamento do algoritmo implementado.**

***Palavras-Chave*— Denoising, DWT, MODWT, FPGA.**

***Abstract*— This paper presents the FPGA implementation of a noise reduction algorithm for audio signals, using variations of the Discrete *Wavelet* Transform (DWT). Initially, the *denoising* algorithm considering the DWT/MODWT is simulated in MATLAB to obtain the different parameters that result in best performance. Next, the resulting algorithm is synthesized on an FPGA, and its behavior is evaluated in simulation. Finally, its performance is evaluated using a development kit. The experimental results obtained demonstrate the good functioning of the implemented algorithm.**

***Keywords*— Denoising, DWT, MODWT, FPGA.**

I. Introdução

Nas últimas décadas, a transformada *wavelet* vem sendo utilizada em diferentes aplicações, tais como: compressão de imagens [1], detecção de sinais [2], reconhecimento de padrões [3] e redução de ruídos (*denoising*) [4]. Normalmente, tais aplicações são implementadas em um computador de propósito geral utilizando MATLAB, Python ou linguagem C. Os avanços na tecnologia, a redução no custo de dispositivos e a diversidade de aplicações utilizando *wavelets* têm motivado sua implementação em dispositivos FPGA [5]–[7].

Neste artigo, é apresentada a implementação em FPGA de um algoritmo para redução de ruídos em sinais de áudio, utilizando a variação *Maximal Overlap* DWT (MODWT) da Transformada *Wavelet* Discreta (DWT) [8]. Inicialmente, diferentes combinações do algoritmo de *denoising* utilizando *wavelets* são implementadas/avaliadas em MATLAB. Na sequência, o algoritmo que apresenta o melhor resultado é selecionado. Finalmente, tal algoritmo é customizado e implementado no FPGA do kit DE2-115 [9].

O restante do artigo está organizado da seguinte forma. A Seção 2 apresenta uma visão geral do algoritmo de denoising considerado, descrevendo também as características dos blocos que compõe tal algoritmo. A Seção 3 apresenta a avaliação e o comportamento dos algoritmos de *denoising* implementados em MATLAB. A Seção 4 apresenta a implementação do algoritmo em FPGA. Resultados experimentais são apresentados e discutidos na Seção 5. Finalmente, a conclusão é apresentada na Seção 6.

Matheus Oliveira, Walter Gontijo, Sidnei Noceti Filho e Eduardo L. O. Batista, LINSE Laboratório de Circuitos e Processamento de Sinais, Departamento de Engenharia Elétrica e Eletrônica, Universidade Federal de Santa Catarina, UFSC, Florianópolis-SC, Brasil, e-mails: matheusaop09@linse.ufsc.br, walter@linse.ufsc.br, sidnei@linse.ufsc.br e eduardo.batista@ufsc.br.

II. Visão Geral

A Fig. 1 mostra o diagrama de blocos para a implementação do algoritmo de redução de ruídos em sinais de áudio, utilizando DWT ou MODWT. Observa-se em tal figura os três blocos principais: transformada direta (T-dir), bloco de *thresholding* (Th) e transformada inversa (T-inv).

A transformada direta (análise) é composta por um par de filtros passa-alta (G) e passa-baixa (H), que decompõe o sinal original em diversos níveis. No bloco *thresholding* (Th), são definidos o método de cálculo do *threshold* e a regra de *thresholding*.

O bloco da transformada inversa (síntese), mostrada na Fig. 1, é também composto por um par de filtros passa-alta ($\tilde{G}$) e passa-baixa ($\tilde{H}$) que recompõe o sinal. Os coeficientes dos filtros são obtidos a partir da definição da *wavelet* mãe.

Na Fig.1, X é dado pela adição de um sinal limpo e um ruido, enquanto Y corresponde ao sinal reconstruído na saída do *denoising*.

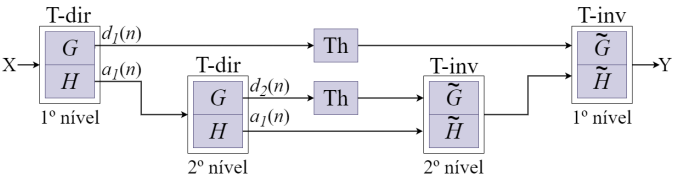

Fig. 1. Diagrama de blocos do algoritmo.

A. DWT

A estrutura de implementação da *wavelet* ilustrada na Fig. 1 é baseada no algoritmo piramidal, originalmente proposto por Mallat [10]. Esse algoritmo considera um par de filtros passa-baixa (H) e passa-alta (G), denominados filtros de síntese, os quais são relacionados por [5]

$$g(n) = (-1)^{n+1}\, h(N - n - 1) \tag{1}$$

onde $g(n)$ e $h(n)$ são as respostas ao impulso de G e H, respectivamente, e N é o número de coeficientes. Esses coeficientes são obtidos a partir da definição do *kernel wavelet* e da função de escala.

A saída do filtro passa-alta é denominada *detail* $d_j(n)$ e a do passa-baixa é denominada *aproximation* $a_j(n)$, sendo elas dadas respectivamente por [5]:

$$d_{j+1}(n) = \sum_k a_j(k)\, g(2n-k) \tag{2}$$

e

$$a_{j+1}(n) = \sum_k a_j(k)\, h(2n-k) \tag{3}$$

onde a_j é a entrada no nível j. O cálculo da transformada inversa (IDWT) também utiliza um par de filtros passa-baixa ($\tilde{H}$) e passa-alta ($\tilde{G}$), denominados filtros de reconstrução. Os coeficientes dos filtros de análise e reconstrução são relacionados por [5]:

$$\tilde{g}(n) = g(N-n-1) \tag{4}$$

e

$$\tilde{h}(n) = h(N-n-1). \tag{5}$$

O sinal de saída reconstruído por esses filtros é dado por [5]

$$a_j(n) = \sum_k d_{j+1}(k)\, \tilde{g}(n-2k) + a_{j+1}(k)\, \tilde{h}(n-2k). \tag{6}$$

B. MODWT

A MODWT é uma modificação da DWT que possibilita melhorar o desempenho de aplicações de *denoising* em tempo real. Por exemplo, a MODWT é bem definida para todos os tamanhos de janelas, ao contrário da DWT, que só é bem definida em janelas que tenham o tamanho expresso em potência de dois. Além do que, ao contrário da DWT, a MODWT não envolve a operação de decimação, sendo assim ela não perde informação durante a decomposição e o sinal decomposto é melhor correlacionado com o sinal original no domínio do tempo.

Em [8], o algoritmo piramidal da MODWT é apresentado, sendo semelhante ao algoritmo piramidal proposto por Mallat [10] para a DWT. Assim como a DWT, a MODWT possui filtros passa-alta ($G*$) e passa-baixa ($H*$) que se relacionam por [8]

$$g* = \frac{g}{\sqrt{2}},\; h* = \frac{h}{\sqrt{2}} \tag{7}$$

onde $g*$ e $h*$ são as respostas ao impulso de $G*$ e $H*$, respectivamente.

C. Denoising por Thresholding

Denoising por *Thresholding* é uma técnica de redução de ruídos que consiste em calcular um parâmetro *threshold* e aplicar uma função nos coeficientes d_j. Tal função tem o objetivo de alterar os coeficientes d_j de forma a reduzir o ruído no sinal de saída reconstruído.

Um dos métodos para calcular o valor do *threshold* é o *Universal Threshold*, proposto por Donoho and Johnstone [11], [12]. Nesse método, o *threshold* é dado por

$$\lambda = \frac{mediana(|d_j|)}{0,6745}\,\sqrt{2\,\log(N)} \tag{8}$$

onde N é o número de amostras do sinal ruidoso. A partir do valor do *threshold* λ, os coeficientes d_j são avaliados por uma regra de *threshold*. Algumas das regras disponíveis na literatura são: *soft thresholding* ($T_{\rm hs}$) e *hard thresholding* ($T_{\rm hh}$), sendo elas definidas como

$$T_{\rm hs}(d_j,\lambda) = \begin{cases} 0, & \text{se } |d_j| \le \lambda \\ d_j - \lambda, & \text{se } d_j > \lambda \\ d_j + \lambda, & \text{se } d_j < \lambda \end{cases} \tag{9}$$

e

$$T_{\rm hh}(d_j,\lambda) = \begin{cases} d_j, & \text{se } |d_j| > \lambda \\ 0, & \text{se } |d_j| \le \lambda. \end{cases} \tag{10}$$

III. Implementação em Matlab

Neste trabalho, são utilizadas as funções "wden" e "wdenoise", que implementam o algoritmo de denoising em sinais unidimensionais e estão disponíveis na versão R2018 do MATLAB [13], [14]. A partir dessas funções, são avaliados diferentes cenários, tais como: tipo da transformada *wavelet* (DWT, MODWT), função *wavelet* (Symlet, Daubechies e Coiflet), nível de decomposição (1 a 5), método de cálculo de *threshold*, regra de *thresholding* e dependência do *threshold* ao nível. Os métodos de cálculo de *threshold* considerados são: Minimax (M), Sure (S), *Universal threshold* (U), *False Discovery Rate* (FDR), *Block* James-Stein (BJS), *Empirical Bayes* (B) e Heursure (H). Já as regras de *thresholding* consideradas são: *Soft*, *Hard*, Média, Mediana e James-Stein. A Fig. 2 mostra o fluxograma para a avaliação dos diferentes cenários.

A partir dos experimentos realizados, foi constatado que os parâmetros de maior influência no desempenho do algoritmo são: método de *threshold*, regra de *threshold* e dependência do *threshold* ao nível. Tais resultados estão de acordo com [15].

IV. Implementação em FPGA

O algoritmo de *denoising* foi implementado na linguagem VHDL com o ambiente QUARTUS [16] no dispositivo EP4CE115F29C [9]. A entrada e a saída de dados são de 16 bits e os cálculos internos são realizados com precisão de 16 e 32 bits.

De acordo com os resultados obtidos em MATLAB, a implementação em FPGA será composta por: transformada MODWT/IMODWT com método do tipo *universal threshold*, regra de *threshold soft*/*hard*, 5 níveis de decomposição e dependência do *threshold* ao nível. Para realizar tal implementação e possibilitar que o algoritmo seja executado em tempo real, são necessários ajustes/customizações, os quais são detalhados na sequência.

A. Tranformadas MODWT/IMODWT

A implementação em FPGA de tais transformadas utiliza as mesmas estratégias apresentadas em [5] para a implementação da DWT. Particularmente, é utilizada a forma transposta para implementar o filtro FIR das transformadas, dado que essa forma mostra-se como mais eficiente em FPGA [17].

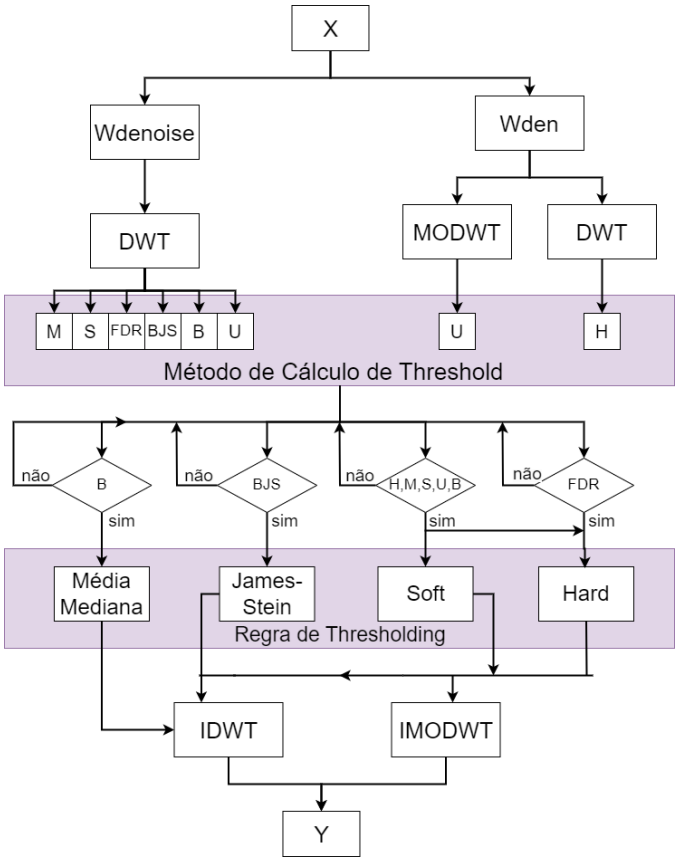

Fig. 2. Fluxograma da avaliação do algoritmo em MATLAB.

Na Fig. 3, é mostrado o diagrama em blocos da forma transposta de um filtro FIR, adaptado para a implementação da MODWT. As multiplicações da entrada x(n) pelos coeficientes f_L são realizadas em paralelo, atrasadas por registradores e depois somadas sucessivamente. Os coeficientes f_L são quantizados em 16 bits e armazenados em registradores. O atraso k (adaptação da implementação do filtro FIR para a MODWT) é sintetizado via *shift registers* e seu valor é dado por

$$k = 2^{j-1} \tag{11}$$

onde j é o nível de decomposição.

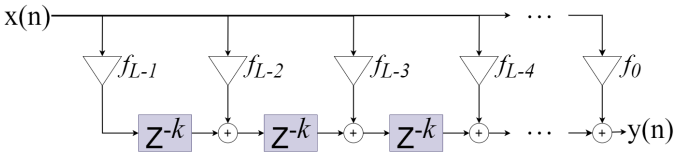

Fig. 3. Diagrama em blocos para implementação da MODWT.

B. Cálculo do Threshold

O cálculo do *threshold* pelo método *Universal Threshold* é apresentado na equação 8. Nas funções "wden" e "wdenoise" do MATLAB R2018 [13], [14], tal método é modificado para que seja dependente do nível de decomposição. O novo método pode ser definido como

$$\lambda_j = \frac{\sqrt{2}\, mediana(|d_j - mediana(d_j)|)}{0,6745} \sqrt{\frac{\log(N)}{2^{j-1}}} \tag{12}$$

onde o índice j corresponde ao nível de decomposição. A implementação em FPGA do cálculo do *threshold* é dada por

$$\lambda_j = \text{média}(|\text{média}(d_j) - d_j|)\, c_j. \tag{13}$$

Comparando as equações 12 e 13, observa-se que a mediana é substituída pela média e os demais termos são definidos por constantes c_j, que são pré-calculadas, quantizadas em 16 bits e armazenadas em registradores. As constantes c_j são dadas por

$$c_j = \frac{\sqrt{2}}{0,6745} \sqrt{\frac{\log(N)}{2^{j-1}}}. \tag{14}$$

As médias foram calculadas através da técnica *exponential smoothing* [18], obtida por

$$m(n) = (1 - \alpha) \cdot m(n-1) + \alpha \cdot d_j(n) \tag{15}$$

onde α é uma constante de tempo. Essa técnica possibilita que o cálculo da média seja realizado de forma recursiva, permitindo sua execução em tempo real. Na Fig. 4, é mostrado o comportamento da média recursiva (linha contínua), comparada com a média teórica (linha pontilhada), para um $\alpha = 0,001$. Observa-se em tal figura que o cálculo recursivo converge para o valor teórico.

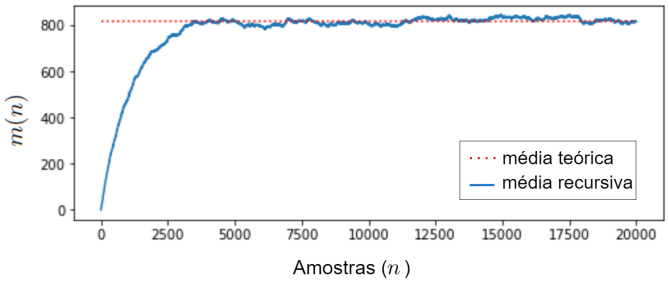

Fig. 4. Comparação entre a média recursiva e o cálculo teórico.

C. Arquitetura da Implementação em FPGA

A arquitetura da implementação em FPGA é ilustrada na Fig. 5 com a apresentação dos principais blocos: MODWT/IMODWT, método de cálculo de *threshold*, regra de *thresholding* e o *delay*. O bloco de *delay* é implementado com *shift registers*, sendo utilizado para sincronizar os coeficientes d_j e a_j, permitindo a reconstrução da saída do algoritmo de *denoising*.

V. Resultados

Esta seção é dedicada à apresentação dos resultados obtidos. Inicialmente, são apresentados os resultados do desempenho do algoritmo em MATLAB para diferentes cenários. Na sequência, são apresentados os resultados da síntese em FPGA e de desempenho em simulação. Finalmente, é avaliado o desempenho do algoritmo ao ser executado no kit de desenvolvimento DE2-115 [9].

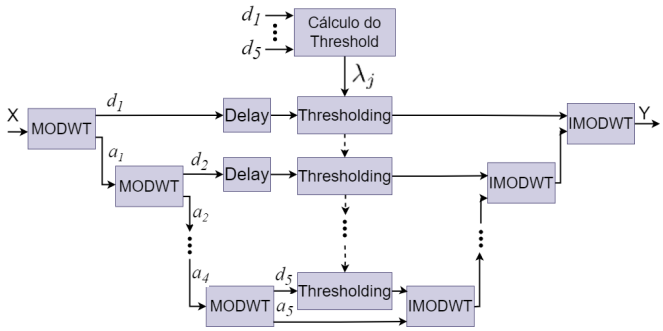

Fig. 5. Arquitetura da implementação em FPGA.

Para a avaliação objetiva do desempenho do algoritmo, é utilizada a SNR (*Signal to Noise Ratio*) de saída, dada por

$$\text{SNR saída (dB)} = 10 \cdot \log_{10}\left(\frac{\sum_{i=1}^{n} s_i^2}{\sum_{i=1}^{n}(s_i - y_i)^2}\right) \qquad (16)$$

onde s é o sinal limpo, y é a saída do algoritmo de *denoising* e n é o numero de amostras.

A. Resultados da Avaliação em MATLAB

Resultados experimentais obtidos em MATLAB mostraram que a *wavelet* db5 (com um total de 20 coeficientes, sendo 10 de *aproximation* e 10 de *detail*) e 5 níveis de decomposição resultaram no melhor desempenho do algoritmo. A partir desse resultado, foram avaliadas combinações de diferentes cenários. Os resultados de desempenho para esses cenários, considerando quatro sinais de áudio com um ruído branco gaussiano e SNR de 5 dB, são apresentados na Tabela I.

A coluna "Cenários" da Tabela I corresponde a: Identificação do tipo da transformada wavelet [Transformada *Wavelet* Discreta (DWT), *Maximal Overlap* DWT (MOD)], método de cálculo de *threshold* utilizado [Minimax (M), Sure (S), *universal threshold* (U), *False Discovery Rate* (FDR), *Block* James-Stein (BJS), *Empirical Bayes* (B) e Heursure (H)], regra de *thresholding* utilizada [*Soft* (ST), *Hard* (HD), Média (ME), Mediana (MD), James-Stein (JS)] e, por último, se o *threshold* é ou não dependente ao nível [Dependente (D), Independente (I)].

Observam-se, em destaque na Tabela I, os maiores valores de SNR para cada sinal de entrada. Também observa-se que o cenário que obteve o melhor desempenho foi o **MOD-U-HD-D**: transformada MODWT com método *universal threshold*, regra de *threshold* do tipo *hard* e dependência do *threshold* ao nível. Esse resultado está de acordo com o apresentado em [19] e é um dos cenários escolhidos para implementação em FPGA. O outro cenário considerado para a implementação é o **MOD-U-ST-D**, alterando apenas a regra de threshold para o tipo *soft*.

B. Resultados da Síntese e Simulação - FPGA

O dispositivo FPGA alvo usado neste trabalho é o EP4CE115F29C7 da família Cyclone IV E. A síntese do algoritmo de *denoising* para o FPGA foi realizada usando a ferramenta Quartus Prime da Intel [16], enquanto as simulações foram realizadas usando o ModelSim [20]. Os resultados obtidos com a síntese são sumarizados na Tabela II. O algoritmo proposto foi implementado no dispositivo alvo utilizando os seguintes recursos: 6% dos pinos, 19% de elementos lógicos e menos de 1% da capacidade total de memória. A frequência máxima obtida para sinal de clock foi de 94,72 MHz.

Os resultados da SNR na simulação realizada no ModelSim [20] são apresentados na Tabela III. Observa-se em tal tabela que a SNR obtida na simulação é equivalente à do MATLAB.

Alguns fatores presentes na implementação em FPGA justificam a diferença da SNR obtida em MATLAB e em simulação. O primeiro fator é o cálculo do *threshold* (λ_j), que em MATLAB é obtido sobre uma única janela e, em FPGA, amostra a amostra. Um segundo fator é a precisão numérica, já que em MATLAB são realizadas operações matemáticas em ponto flutuante e, em FPGA, em ponto fixo.

TABELA I

SNR de saída para diferentes entradas com ruído branco gaussiano e SNR de 5 dB - Resultados em MATLAB

Cenários	Guitarra	Bateria	Voz	Música
DWT-U-ST-I	15,1279	9,0793	11,497	9,2350
DWT-U-ST-D	14,7981	8,5231	10,6236	8,1771
DWT-U-HD-I	15,7289	10,3453	12,6423	10,3799
DWT-U-HD-D	15,3792	9,6558	11,4337	8,3070
MOD-U-ST-D	**18,0346**	11,5950	14,9186	11,6377
MOD-U-HD-D	16,8152	**13,4450**	**15,2471**	14,3172
DWT-H-ST-I	15,3428	9,3764	11,8618	9,505
DWT-H-ST-D	14,9887	8,8006	10,8884	8,2173
DWT-H-HD-I	15,9625	10,6664	13,1098	10,7773
DWT-H-HD-D	15,5958	10,0594	11,8351	8,4316
DWT-S-ST-I	17,1163	12,623	14,7451	**14,4102**
DWT-S-ST-D	16,7860	12,1635	14,1714	11,7326
DWT-S-HD-I	15,1159	9,6098	12,0416	12,6177
DWT-S-HD-D	15,4569	10,2904	12,5912	12,7510
DWT-M-ST-I	15,5602	9,8786	12,226	9,6993
DWT-M-ST-D	15,1796	9,2555	11,1745	8,2566
DWT-M-HD-I	16,0505	11,2436	13,4489	11,0622
DWT-M-HD-D	15,6793	10,6264	12,2564	8,5443
DWT-BJS-JS-I	16,4104	11,7245	14,0223	12,5531
DWT-B-MD-I	16,6448	12,1005	14,3882	13,9143
DWT-B-MD-D	16,1168	11,3060	13,3319	9,7788
DWT-B-ME-I	17,0095	12,6572	14,6892	14,2877
DWT-B-ME-D	16,5075	11,9737	13,9226	10,6802
DWT-B-ST-I	16,2308	11,0598	14,0413	13,4932
DWT-B-ST-D	15,4994	9,9534	12,2473	8,9236
DWT-B-HD-I	16,4126	11,7555	14,0580	13,6056
DWT-B-HD-D	16,1044	11,3374	13,5123	9,9395
DWT-FDR-HD-I	16,3704	11,6804	14,0274	13,0459
DWT-FDR-HD-D	16,0196	11,1080	13,0525	9,5275

TABELA II

Ocupação no FPGA e frequência máxima

Elementos Lógicos	21,388 / 114,480 (19 %)
Pinos	34/529(6%)
Memória(bits)	31,340/3,981,312(<1%)
Frequência Máxima	94,72 MHz

C. Analise de Tempo Real no Kit DE2-115 - FPGA

Na Fig. 6, são mostradas as formas de onda da entrada e da saída adquiridas no CODEC de áudio do kit DE2-115 [9]. A entrada é um sinal de áudio com um ruído branco gaussiano

TABELA III

SNR de saída para diferentes entradas com ruído branco gaussiano e SNR de 5 dB - Resultados da simulação RTL

Cenários	Guitarra	Bateria	Voz	Música
MOD-U-ST-D	17,8385	10,2561	14,4934	10,7770
MOD-U-HD-D	17,2129	11,2985	15,2246	13,1925

e SNR de 5 dB. Pode-se observar em tal figura que na saída o ruído foi removido e as características do áudio original foram mantidas. Fazendo uma análise subjetiva da qualidade do áudio de saída, percebe-se que foi reduzido o ruído de entrada sem distorcer o sinal de interesse.

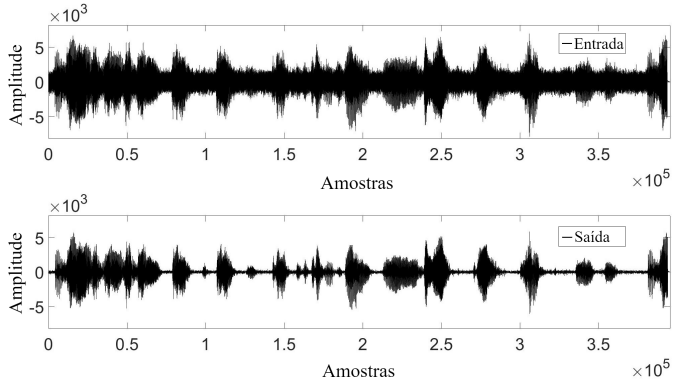

Fig. 6. Formas de onda de entrada e saída no Kit DE2-115, para SNR de 5 dB (entrada) e ruído gaussiano.

Em relação ao tempo de processamento, observou-se um atraso de 292 amostras (6,08 milissegundos), devido aos estágios de *pipeline* presentes na implementação. Observou-se ainda que o cálculo da média recursiva leva em torno de 1 segundo para convergir para o valor do *threshold*.

Considerando que a frequência de amostragem para a aquisição do sinal de áudio é de 48 kHz e o processamento é realizado amostra a amostra, conclui-se que a frequência miníma de operação do algoritmo é de 48 kHz. Assim, considerando que a frequência máxima de operação obtida na síntese é de 94,7 MHz, conclui-se que o *denoising* implementado em FPGA permite realizar o processamento do sinal em tempo real.

VI. Comentários e Conclusões Finais

Neste trabalho, foram discutidas implementações em MATLAB e em FPGA de um algoritmo de *denoising* utilizando a MODWT. Inicialmente, foi avaliado o desempenho considerando diferentes parâmetros, tais como: tipo da transformada *wavelet*, função *wavelet*, nível de decomposição, método de *threshold*, regra de *threshold* e dependência do *threshold* ao nível. Observou-se que o melhor desempenho foi obtido para: MODWT, db5, 5 níveis, o método do tipo *universal threshold*, regra do tipo *soft*/*hard* e dependência do *threshold* ao nível. Na sequência, o algoritmo resultante foi customizado e sintetizado no FPGA EP4CE115F29C7 do kit DE2-115. Finalmente, o desempenho do algoritmo foi avaliado em tempo real, confirmando seu excelente funcionamento. Os resultados obtidos mostram o desempenho adequado da implementação do algoritmo de *denoising* considerado.

Referências

[1] R. Krishnaswamy and S. NirmalaDevi, "Efficient medical image compression based on integer wavelet transform," in *Proc. 2020 Sixth International Conference on Bio Signals, Images, and Instrumentation (ICBSII)*, Chennai, India, Feb. 2020, pp. 1–5.

[2] T. H. Tran, "Improvement of methods for detection of characteristic points of biomedical signals based on continuous wavelet transformation in the continuous flow of data," in *Proc. 2018 Third International Conference on Human Factors in Complex Technical Systems and Environments (ERGO)s and Environments (ERGO)*, Saint Petersburg, Russia, Jul. 2018, pp. 189–192.

[3] S. Pittner and S. Kamarthi, "Feature extraction from wavelet coefficients for pattern recognition tasks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 1, pp. 83–88, Jan. 1999.

[4] J. Kim, C. Y. Chun, and B. H. Cho, "Evaluation of noise reduction in experimental battery pack voltage using discrete wavelet transform and wavelet packet transform," in *Proc. 2015 IEEE International Telecommunications Energy Conference (INTELEC)*, Osaka, Japan, Oct. 2015, pp. 1–5.

[5] M. Bahoura and H. Ezzaidi, "Fpga-implementation of discrete wavelet transform with application to signal denoising," *Circuits Systems and Signal Processing*, vol. 31, pp. 987–1015, Jun. 2012.

[6] A. Sahour, F. Boumehrez, H. Djellab, and F. Maamri, "Dwt denoising signal implementation based on fpga target," in *Proc. 2023 IEEE International Conference on Advanced Systems and Emergent Technologies (IC ASET)*, Hammamet, Tunisia, Jun. 2023, pp. 1–6.

[7] P. Goel, M. Chandra, A. Anand, and A. Kar, "An improved wavelet-based signal-denoising architecture with less hardware consumption," *Applied Acoustics*, vol. 156, pp. 120–127, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0003682X19304827

[8] D. Percival and A. Walden, *Wavelet Methods for Time Series Analysis*, ser. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2000.

[9] TerasIC, "DE2-115 Development and Education Board," https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=502.

[10] S. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, pp. 674–693, Jul. 1989.

[11] D. L. Donoho, I. M. Johnstone *et al.*, "Ideal denoising in an orthonormal basis chosen from a library of bases," *Comptes rendus de l'Académie des sciences. Série I, Mathématique*, vol. 319, no. 12, pp. 1317–1322, 1994.

[12] D. L. Donoho and I. M. Johnstone, "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol. 81, no. 3, pp. 425–455, 1994. [Online]. Available: http://www.jstor.org/stable/2337118

[13] MathWorks Company, "wden," https://www.mathworks.com/help/wavelet/ref/wden.html.

[14] MathWorks, "wdenoise," https://www.mathworks.com/help/wavelet/ref/wdenoise.html#mw_76f9b236-67c7-4c95-b3ac-4d962dbcd7eb.

[15] S. Mihov, R. Ivanov, and A. Popov, "Denoising speech signals by wavelet transform," *Annual Journal of Electronics*, pp. 1313–1842, Jan. 2009.

[16] Intel FPGA, "Quartus Prime Sofwtware," https://www.intel.com/content/www/us/en/software-kit/665990/intel-quartus-prime-lite-edition-design-software-version-18-1-for-windows.html.

[17] U. Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays*, ser. Signals and Communication Technology. Springer Berlin Heidelberg, 2013. [Online]. Available: https://books.google.com.br/books?id=MBz2CAAAQBAJ

[18] E. S. Gardner, "Exponential smoothing: The state of the art—part ii," *International Journal of Forecasting*, vol. 22, no. 4, pp. 637–666, Oct. 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207006000392

[19] I. K. Alak and S. Ozaydin, "Speech denoising with maximal overlap discrete wavelet transform," in *Proc. 2022 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, Ras Al Khaimah, United Arab Emirates, Nov. 2022, pp. 27–30.

[20] I. FPGA, "ModelSim Standart Edition Software," https://www.intel.com/content/www/us/en/software-kit/750368/modelsim-intel-fpgas-standard-edition-software-version-18-1.html.