Relação custo benefício *hardware/software* de um filtro de resposta finita implementado em *Field-Programmable Gate Arrays*

Gustavo Farhat de Araujo e Roberto d'Amore
Instituto Tecnológico da Aeronáutica - Pça. Marechal Eduardo Gomes, 50 - Vila das Acácias - CEP 12.228-900— São José dos Campos/SP - Brasil

***Resumo* – Este trabalho aborda quatro implementações de um filtro FIR em um FPGA, com o objetivo de se estudar a relação custo benefício *hardware/software*, considerando a velocidade de processamento e o uso de recursos de um FPGA. Instruções dedicadas foram incluídas no conjunto de instruções disponíveis do processador Nios sintetizado no FGPA para estudo do desempenho. Foi analisado um filtro FIR *windowed-sinc* passa-baixas com uma janela Blackman. A partir de um sinal de entrada composto, foi gerado um sinal de saída filtrado e verificado o tempo de processamento e o percentual de recursos utilizados por cada implementação. Na implementação com mais recurso de *hardware*, o tempo de execução foi de 40µs enquanto no modelo implementado exclusivamente por *software*. o tempo necessário atingiu 14ms. Nas implementações onde se predominou o uso de *hardware* foram empregados 70% dos recursos do FPGA, contra 4% no caso do algoritmo executado predominantemente em *software*.**

***Palavras-Chave* – Integração de Sistemas Embarcados, FPGA, Nios.**

I. INTRODUÇÃO

Um fato comumente observado em todos os conflitos armados da história da humanidade é a busca constante pela superação tecnologicamente frente ao inimigo. Nas guerras mais recentes, armamentos inteligentes contribuíram essencialmente para as conquistas e vitórias almejadas.

A aplicação operacional da Guerra Eletrônica (GE), cujo objetivo é a obtenção de vantagens sobre o inimigo mediante a exploração do espectro eletromagnético, destaca-se pelo uso de sistemas tais como: radares de vigilância, radares diretores de tiro, radares imageadores, *Radar Warning Receivers* (RWR), *Global Position Satellites* (GPS), sistemas de comunicação e controle (C2), mísseis infravermelho, mísseis antirradiação, etc.

Para proporcionar superioridade no combate, esses sistemas de GE têm em comum a necessidade de processar sinais digitais em altíssima velocidade, de maneira que o tempo de resposta do armamento seja cada vez menor.

Além disso, um bom aproveitamento do espaço físico é de fundamental importância para o emprego em plataformas com restrições desse tipo, tais como mísseis e Veículos Aéreos Não Tripulados (VANT).

Gustavo Farhat de Araujo, gustavo.farhat@gmail.com, Tel +55-12-982649806, Roberto d'Amore, damore@ita.br, Tel +55-12-39476876
Este trabalho foi apoiado pelo Comando da Aeronáutica, através do Programa de Pós-Graduação em Aplicações Operacionais (PPGAO).

Sendo assim, há uma crescente demanda pelo desenvolvimento de unidades de processamento de sinais digitais, comuns aos sistemas de GE, mais rápidas e de dimensões reduzidas.

Uma das formas se obter ganho em velocidade de processamento é o uso de *hardware* dedicado em tarefas específicas. Entretanto, o acréscimo de um novo *hardware* tem um impacto no custo que deve ser analisado em face aos requisitos a serem atendidos.

O uso de FPGA (*Field-Programmable Gate Arrays*) aparece nesse cenário como uma solução viável para o problema, uma vez que possibilita altas velocidades de processamento em uma área reduzida de *hardware*. A flexibilidade inerente dos FPGAs permite que sejam sintetizados microprocessadores e periféricos, personalizados de acordo com a aplicação desejada, no mesmo dispositivo.

Esta técnica foi explorada em vários trabalhos. Nekoei, e Kavian [1] compararam implementações de um filtro FIR em *hardware*. Em [2, 3, 4] foi estudada a velocidade de processamento para rotinas executadas via *software* e/ou *hardware*. Cardarilli, Nunzio e Fazzolari analisaram algoritmos de criptografia [2], Boudabous *et al.* rotinas de redução de ruído de imagens [3], e Zhao, Zhang e Lin a codificação de sinais vocais (*vocoder*) [4]. Um estudo com foco em consumo de potência foi feito em [5]. Todos esses trabalhos convergiram para uma mesma conclusão: atribuindo-se ao *hardware* uma carga maior de processamento, os algoritmos são executados com maior velocidade.

Os resultados desses trabalhos motivaram o estudo de uma aplicação baseada em FPGA para processamento de sinais. O objetivo desse estudo foi quantificar o desempenho de um módulo típico de um sistema de processamento de sinais, sob diferentes técnicas de implementação. Essas técnicas diferenciam-se pela atribuição de maior ou menor carga de processamento ao *hardware* ou ao *software* do processador. Assim, pretendeu-se obter com os resultados a relação custo/benefício de cada técnica implementada em função da velocidade de processamento e a quantidade de recursos utilizados.

Para se colocar a prova as diferentes técnicas de implementação, foi necessário empregar um algoritmo, que além de comum aos sistemas de processamentos de sinais, demandasse um número elevado de iterações matemáticas com certa complexidade para computação numérica. Por ser um filtro que possui essas características, o filtro FIR (*Finite Impulse Response*) passa baixas foi então escolhido e utilizado neste trabalho.

O algoritmo de um filtro FIR com M termos realiza M multiplicações e M acumulações para cada valor de saída. O

sinal de entrada é amostrado com *N* amostras. Sendo assim, são realizadas *N*x*M* multiplicações e *N*x*M* acumulações. Essa grande quantidade de operações, que impacta consideravelmente no processamento, foi explorada nesse trabalho, para se determinar as vantagens e desvantagens das diferentes implementações por *hardware*/*software*. Quatro implementações foram analisadas.

II. FILTRO E PLATAFORMA DE TESTES

Tomou-se como referência para o projeto do filtro uma entrada de dados proveniente de um conversor analógico-digital de 16 bits disponível no mercado para aplicações em processamento de sinais de alta velocidade: LTC-2389-16 fabricado pela *Linear Technology*, com frequência de amostragem de 2,5MHz.

O filtro em estudo supõe uma frequência de amostragem de 2,5 MHz. Considerando o teorema de Nyquist, que estabelece, como frequência mínima de amostragem, o dobro da maior frequência do sinal amostrado, no caso em questão, pode-se reconstruir sinais de até 1,25Mhz. Para se obter um sinal com menor distorção, optou-se por estabelecer a frequência de corte do filtro em aproximadamente um quarto da frequência de amostragem: 600KHz.

Considerando uma limitação de 133 *embedded multipliers* da plataforma FPGA utilizada para testes, optou-se por restringir ao *hardware* o uso de 100 *embedded multipliers*. Assim limitou-se o total de coeficientes do filtro (*M*) a 100, de maneira que, no máximo, 100 multiplicações pudessem ser realizadas simultaneamente pelo hardware.

Para o filtro rejeitar frequências superiores a 600kHz, foi projetado um filtro FIR passa-baixas, do tipo *Windowed-Sinc,* com frequência de corte (f_c) de 0,24 (24% da frequência de amostragem).

Conforme [6], um filtro de 100 coeficientes (*M*) obtém uma variação de 99% a 1% de amplitude no sinal de saída, em uma banda de transição (*BW*) de 100kHz (4% da frequência de amostragem).

$$M = \frac{4}{BW} \tag{1}$$

Uma janela Blackman (2) que apesar de apresentar uma velocidade de *roll-off* menor que a janela de Hamming, foi empregada por ter como característica uma maior atenuação de corte. Como resultado foram obtidos os valores dos 100 coeficientes (*h*) do filtro, cuja resposta ao impulso pode ser visualizada na Fig. 1.

$$h[i] = K\frac{\sin\left(2\pi f_c\left(i - M/2\right)\right)}{i - M/2}\left[0{,}42 - 0{,}5\cos\left(\frac{2\pi i}{M}\right) + 0{,}08\cos\left(\frac{4\pi i}{M}\right)\right] \tag{2}$$

As amostras do sinal de entrada ($x[i]$) (Fig. 2), similar às utilizadas em [1], foram determinadas por (3), a partir da soma de duas formas de onda do tipo senoidal, uma ($x_1[i]$) com frequência de 500kHz (Fig. 3) e a outra ($x_2[i]$) com frequência de 700kHz (Fig. 4).

$$x[i] = \sin\left(\frac{2\pi f_1 i}{2{,}5x10^6}\right) + \sin\left(\frac{2\pi f_2 i}{2{,}5x10^6}\right) \tag{3}$$

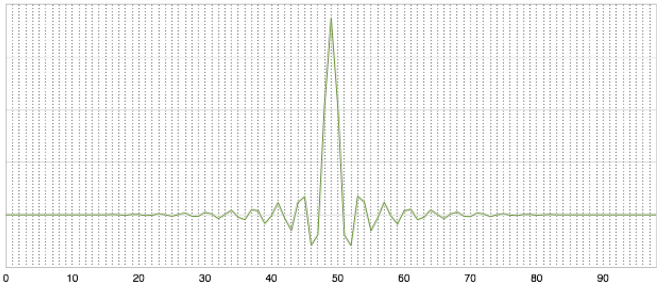

Fig. 1. Resposta do filtro ao impulso: h[i].

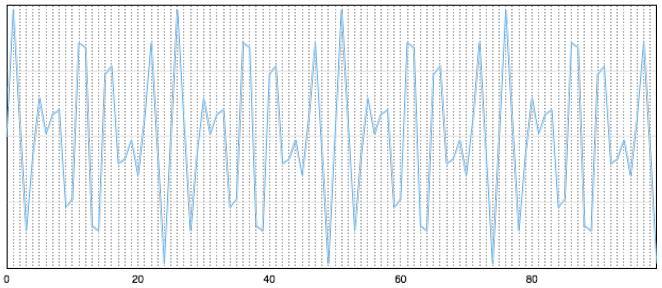

Fig. 2. Sinal de Entrada: $x[i] = x_1[i] + x_2[i]$.

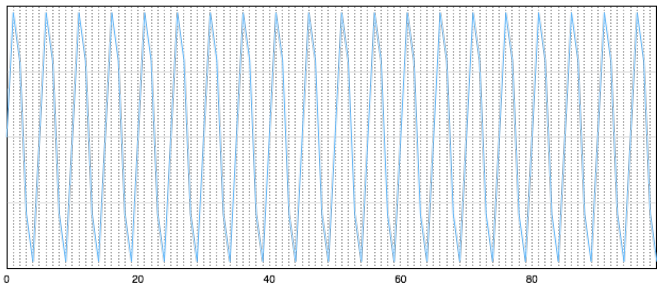

Fig. 3. Sinal $x_1[i]$ (f = 500kHz).

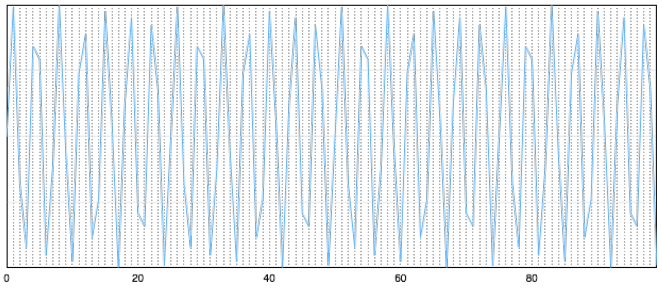

Fig. 4. Sinal $x_2[i]$ (f = 700kHz).

Como plataforma de testes foi utilizado um FPGA Cyclone IV E e o microprocessador embarcado Nios II generation 2 [7] [8].

A Fig. 5 apresenta a personalização feita no sistema microprocessado, configurado com o núcleo do microprocessador Nios II (2KB de cache), uma memória RAM para instruções e dados de 256KB, um *Custom hardware* (utilizando a interface *Custom Instruction* do microprocessador) e um periférico do tipo *Performance Counter* (para medir a quantidade de ciclos empregados para execução dos códigos).

As *Custom Instructions* são executadas por circuitos dedicados integrados à ALU (*Arithmetic and Logic Unit*) do microprocessador. As *Custom Instructions* podem ser projetadas para executar operações simples ou complexas, com o intuito de acelerar parte do código que seria executado por *software*. A Fig. 6 ilustra como o microprocessador trata

as operações realizadas pelas *Custom Instructions* em conjunto com as operações realizadas pela ALU.

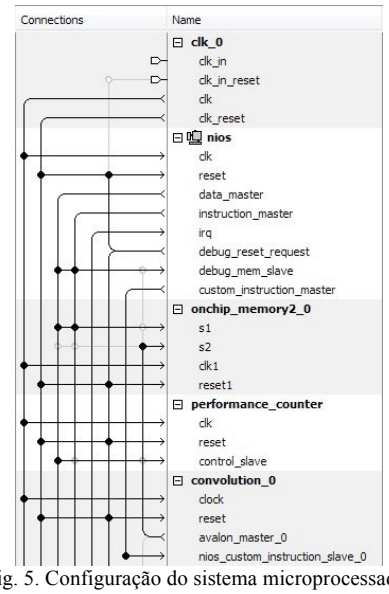

Fig. 5. Configuração do sistema microprocessado.

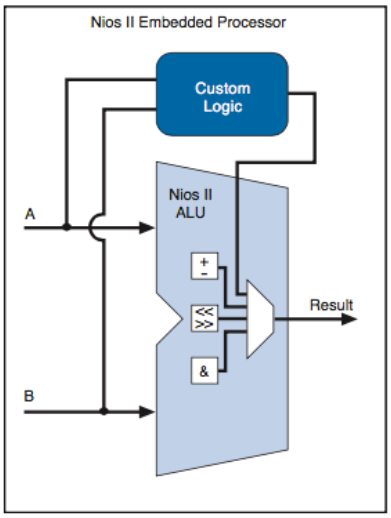

Fig. 6. Integração das *Custom Instructions* à ALU [9].

III. MÉTODO

O algoritmo de um filtro FIR [6], tendo (4) como base para a sua implementação, representa a convolução de dois sinais (entrada e filtro), que produzem o sinal filtrado (sinal de saída).

$$y(i) = \sum_{j=o}^{M-1} h(j) \times x(i-j) \qquad (4)$$

Foram definidas quatro diferentes implementações, aqui denominadas: SW total, HW Multiplicador-Somador, HW Convolução Serial e HW Convolução Paralela.

Na implementação SW total, todo o algoritmo é processado por *software*, ou seja, pelo código desenvolvido e compilado em linguagem C, executado pelo Nios II. As bases desse código são dois *loops* FOR, responsáveis por controlar as multiplicações e acumulações sucessivas, conforme Fig 7.

```
for (i=0;i<N;i++)
{
   y[i] = 0;
   for (j=0;(j<=i) && (j<=M-1);j++)
   {
      if ((i-j>=0) && (i-j<K))
      {
         y[i] = y[i] + x[i-j] * h[j];
      }
   }
}
```

Fig. 7. *Loops* FOR e multiplicações/acumulações via SW

As outras três implementações fizeram uso de um mesmo recurso do Nios II, *Custom Instructions*, já explorado de formas diferentes por [2], [3], [4], [10], [11], [12] e [13].

Na segunda implementação, denominada HW Multiplicador-Somador, manteve-se o *loop* FOR do código em C da implementação SW total, porém foi feito um processamento mínimo em *hardware*, onde as multiplicações e acumulações passaram a ser executadas por uma *Custom Instruction*, conforme Fig. 8.

A *Custom Instruction* foi projetada em VHDL (*Very High Speed Integrated Circuits Hardware Description Language*) para executar cada multiplicação e acumulação em um ciclo de *clock*. O diagrama da Fig. 9 apresenta um diagrama de blocos do módulo de *hardware* projetado para processar as multiplicações e acumulações em auxílio ao *software*.

```
for (i=0;i<N;i++)
{
   for (j=0;(j<=i) && (j<=M-1);j++)
   {
      if ((i-j>=0) && (i-j<K))
      {
         asm ("custom 0, c0, %1, %2" : "=r"
         (y[i]) : "r" (x[i-j]), "r" (h[j]);
      }
   }
   asm ("custom 0, c0, %1, %2" : "=r"
   (y[i]) : "r" (x[i-j]), "r" (h[j]);
}
```

Fig. 8. *Loops* FOR e multiplicações/acumulações via HW

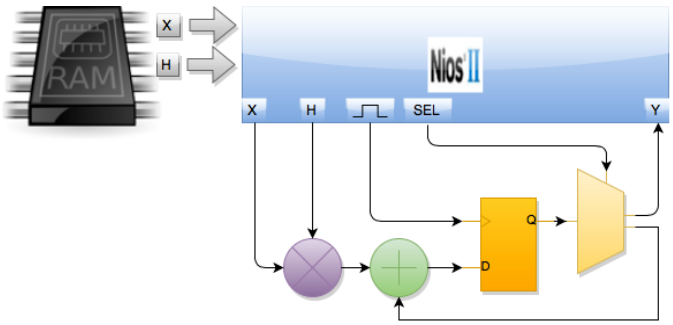

Fig. 9. HW Multiplicador-Somador

Na terceira e na quarta implementação, uma alteração substancial foi realizada. Nessas implementações a memória RAM passou a ser compartilhada entre o Nios e o *hardware* personalizado. Dessa forma, ao invés do *software* controlar as multiplicações e adições, o *hardware* é responsável por essa função. Através da interface *Custom Instruction*, o software passa para o hardware o endereço dos *arrays* de entrada, do filtro e de saída, conforme Fig. 10. A partir desse ponto, controlado por uma máquina de estados, o *hardware*

armazena os dados da memória em seus registradores e executa todo o algoritmo. Cada valor calculado é então gravado no endereço de memória reservado para o *array* de saída.

```
done = ALT_CI_CONVOLUTION(0, &array);
```

Fig. 10. Interface *Custom Instruction* utilizada nas implementações 3 e 4

Na implementação HW Convolução Serial, para cada valor de saída a ser calculado, a máquina de estados executa uma multiplicação e uma adição por ciclo de *clock*. Dessa forma, para efetuar a multiplicação e adição com um filtro de 100 termos, são necessários 100 ciclos de *clock*. Nessa implementação, conforme Fig. 11, é utilizado um único *embedded multiplier*.

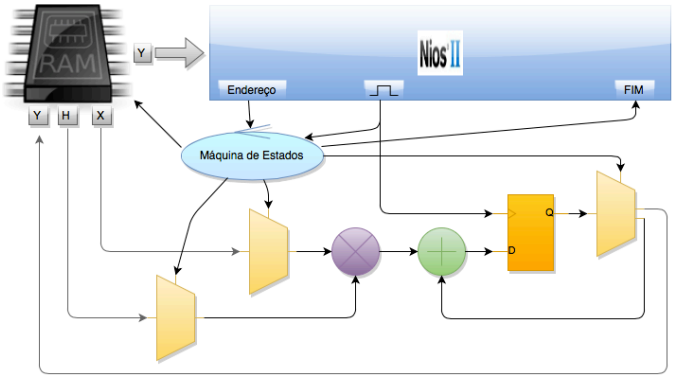

Fig. 11. HW Convolução Serial

Na quarta e última implementação, denominada HW Convolução Paralela, a máquina de estados foi projetada para executar, a cada ciclo de *clock*, todas as 100 multiplicações e adições. Dessa forma, para efetuar a multiplicação e adição com um filtro de 100 termos, seria necessário um único ciclo de *clock*. Entretanto, devido ao circuito combinacional mais complexo, o atraso na propagação dos dados entre registradores é bem elevado, chegando a 290ns. Em função desse atraso, foi inserido um *multi-cycle path* ou seja, forçou-se um atraso de oito ciclos de *clock* para o cálculo de cada valor de saída. Nessa implementação, conforme Fig. 12, foram utilizados 100 (cem) *embedded multipliers*.

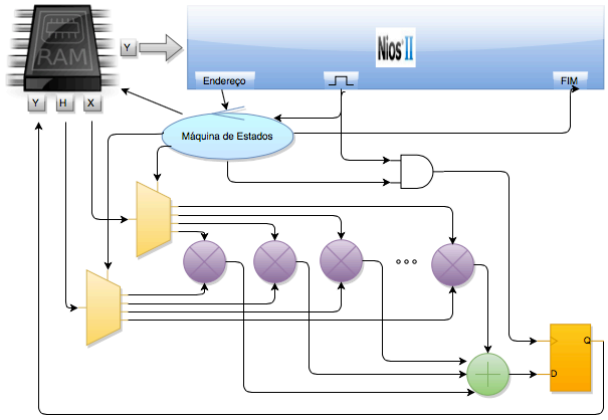

Fig. 12. HW Convolução Paralela

IV. Análise dos Resultados

De acordo com a proposta, as quatro implementações foram testadas, obtendo-se o sinal de saída filtrado de 500kHz, com amplitude de 32 bits, ponto fixo, conforme apresentado na Fig.13.

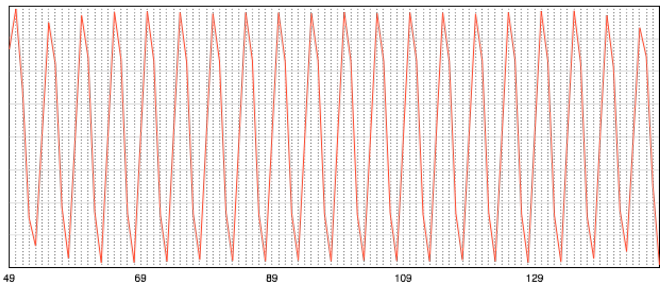

Fig. 13. Sinal de saída y[i] (f = 500kHz)

Seus resultados foram então comparados, com objetivo de se determinar a velocidade com que cada implementação é executada, considerando a frequência máxima de operação (f_{max}) de cada uma delas.

A análise dos resultados também teve como objetivo determinar os recursos do FPGA demandados por cada implementação.

A tabela I apresenta os desempenhos obtidos nas quatro implantações, considerando a quantidade de ciclos utilizados no cálculo e o tempo estimado em função de $f_{max,}$ determinada para cada implementação.

TABELA I TEMPO DE PROCESSAMENTO

	Implementação	Ciclos	f_{max} (MHz)	Tempo total (µs)
1	SW Total	1.210.614	80,80	14.982,85
2	HW Multiplicador-Somador	1.041.996	79,59	13.092,05
3	HW Convolução Serial	21.022	71,03	295,96
4	HW Convolução Paralela	2.432	60,54	40,17

Analisando a tabela I, podemos verificar que a implementação 4 empregou a menor quantidade de ciclos de *clock* no processamento, conforme já antecipado por [13]: aproximadamente um décimo da implementação 3, que ficou em segundo lugar nesse quesito. A implementação 2, apesar da grande quantidade de ciclos de *clock* empregados, comparado às implementações 3 e 4, executou o processamento com uma quantidade bem menor que a utilizada pela implementação 1, apresentando um número aproximadamente 14% menor de ciclos de *clock*.

Vale ressaltar que a complexidade do circuito combinacional da implementação 4 teve como consequência uma maior limitação de f_{max}. Porém, essa limitação de f_{max} não foi suficiente para inviabilizar tal implementação, uma vez que o seu tempo total de execução ficou ainda 7,4 vezes menor que o da implementação 3, 347 vezes menor que o da implementação 2 e 374 vezes menor que o da implementação 1, consideradas as respectivas f_{max}.

É importante observar ainda que a implementação 2 apresentou uma frequência máxima f_{max} similar à da implementação 1, o que confirma a vantagem da implementação daquela sobre esta.

A segunda análise feita diz respeito aos recursos utilizados do FPGA. Dependendo do projeto, é de extrema

importância que algumas considerações sejam feitas em relação aos recursos utilizados, pois o filtro implementado pode ser apenas uma parte de um projeto em que outros componentes de *hardware* necessitam ser implementados. Nesses casos, a decisão quanto a uma determinada implementação não pode ser feita apenas com base na velocidade de processamento. A tabela II apresenta os recursos utilizados do FPGA Cyclone IV E.

Analisando os dados da tabela II, verificamos um maior uso dos recursos do FPGA, quanto menor o tempo de processamento das implementações.

A implementação 4, que apresentou o melhor desempenho, fez uso de aproximadamente 70% dos elementos lógicos e funções combinacionais e de 38,7% dos circuitos multiplicadores disponíveis.

TABELA II RECURSOS UTILIZADOS

	Implementação	Logic Elements (LE) (114.480)	Combina-tional Functions (114.480)	Dedicated Logic Registers (114.480)	Embedded Multipliers (532)
1	SW Total	4,3% (4.968)	4,0% (4.591)	2,5% (2.905)	1,1% (6)
2	HW Multiplicador-Somador	4,4% (5.077)	4,1% (4.676)	2,6% (2.963)	1,5% (8)
3	HW Convolução Serial	8,4% (9.606)	6,6% (7.535)	5,6% (6.450)	1,5% (8)
4	HW Convolução Paralela	73,6% (84.308)	72,3% (82.808)	7% (8.011)	38,7% (206)

A implementação 3, que apresentou um bom desempenho, porém inferior à implementação 4, fez um uso bem menor dos recursos do FPGA, apesar de representar, em alguns casos, aproximadamente o dobro das implementações 1 e 2.

A implementação 2 fez uso de aproximadamente a mesma quantidade de recursos da implementação 1.

V. CONSIDERAÇÕES FINAIS

Com base nos resultados analisados, foi possível identificar a implementação 4, HW Convolução Paralela, como a mais indicada para um projeto que exija grande velocidade de processamento e que não possua restrições significativas de recursos do FPGA.

As implementações 1 e 2 apresentaram um baixíssimo desempenho com relação ao tempo de processamento, uma vez que executam grande parte do algoritmo por *software*. Nas aplicações em que o fator crítico não seja o desempenho, mas sim os recursos disponíveis do FPGA, recomenda-se a implementação 2, devido ao seu menor tempo de processamento, frente a uma mesma quantidade de recursos utilizados.

A implementação 3, HW Convolução Serial, destacou-se como uma solução intermediária, para projetos nos quais a quantidade de recursos disponíveis do FPGA seja um fator crítico. Entretanto sua implementação deve estar condicionada a uma avaliação do impacto do aumento do tempo de processamento em relação à implementação 4.

O estudo permitiu avaliar também a aplicação das *Custom Instructions* como meio para acelerar o processamento em regiões específicas de um código. Regiões que demandem tempo de processamento elevado e o desenvolvimento de um hardware específico para realizar as mesmas operações seja viável, são passiveis de uma análise semelhante da relação custo / benefício.

REFERÊNCIAS

[1] F. Nekoei, Y. S. Kavian, and O. Strobel, “Some schemes of realization digital FIR filters on FPGA for communication applications,” pp. 616–619, 2010.

[2] G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, and M. Re, “TDES cryptography algorithm acceleration using a Reconfigurable Functional Unit,” pp. 419–422, 2014.

[3] A. Boudabous, A. Ben Atitallah, P. Kadionik, and N. Masmoudi, “HW / SW FPGA Implementation of Vector Median Filter,” no. 1, pp. 101–104, 2007.

[4] Y. Zhao, S. Zhang, and X. Lin, “Two methods of Design and Implementation of ACELP Vocoder,” 2013.

[5] W. Wang, M. N. S. Swamy, and M. O. Ahmad, “Low power FIR filter FPGA implementation based on distributed arithmetic and residue number system,” *Proc. 44th IEEE 2001 Midwest Symp. Circuits Syst. MWSCAS 2001 (Cat. No.01CH37257)*, vol. 1, pp. 102–105, 2001.

[6] S. W. Smith, *The scientist and engineer’s guide to digital signal processing*, Second Edi. San Diego, Calif., 1999.

[7] Altera, “Nios II Processor Reference Handbook,” no. February, pp. 1–288, 2014.

[8] Altera, “Nios II Software Developer ’ s,” p. 498, 2009.

[9] Altera Corporation, “Nios II custom instruction user guide,” *Builder*, 2011.

[10] S. M. Vidanagamachchi, S. D. Dewasurendra, and R. G. Ragel, “Hardware Software Co-design of the Aho-Corasick Algorithm : Scalable for Protein Identification ?,” pp. 321–325, 2013.

[11] Y. Tian, Y. Li, Q. Dong, and A. Definition, “Nios II Custom Instruction on FIR Filter Arithmetic,” pp. 135–139, 2012.

[12] A. Ben Atitallah, P. Kadionik, F. Ghozzi, P. Nouel, N. Masmoudi, and H. Levi, “Implementation of the Video Coder in,” no. 1, pp. 814–817.

[13] P. Kollig, B. M. Al-Hashimi, and K. M. Abbott, “FPGA implementation of high performance FIR filters,” *Proc. 1997 IEEE Int. Symp. Circuits Syst. Circuits Syst. Inf. Age ISCAS ’97*, vol. 4, pp. 2240–2243, 1997.