Uma Arquitetura para Sistemas Micropipeline de Alto Desempenho Voltada para FPGAs

Duarte L. Oliveira[1], Kledermon Garcia[2], Roberto d'Amore[1]
[1]Divisão de Engenharia Eletrônica do Instituto Tecnológico de Aeronáutica – ITA – IEEA
[2]Divisão de Sistemas Aeronáuticos do Instituto de Aeronáutica e Espaço – IAE-CTA

***Resumo* — O paradigma assíncrono possui características interessantes devido à ausência do sinal de *clock* e que pode ser uma alternativa de projeto. Este paradigma tem vários estilos de projeto, onde o estilo *micropipeline* é o mais indicado para a plataforma FPGA, devido à simplicidade do seu controle. Neste artigo, propomos uma arquitetura para implementar sistemas digitais assíncronos no estilo *micropipeline bundled-data*. Para implementações em FPGAs, a arquitetura proposta apresenta um melhor desempenho, quando comparada com a arquitetura denominada MOUSETRAP, que é considerada o estado da arte.**

***Palavras-chave*—Lógica assíncrona, FPGA, *micropipeline*, controlador, *Bundled-data*.**

I. INTRODUÇÃO

Sistemas digitais embarcados requerem alta capacidade de integração, alta velocidade e baixo consumo de energia [1,2].

Os dispositivos FPGAs tornaram-se um meio bastante popular para desenvolver e implementar circuitos digitais devido ao custo e tempo de projeto. FPGAs de alto desempenho são implementados na tecnologia MOS *Deep-Sub-Micron* (MOS-DSM). Nessa tecnologia, o atraso nas trilhas deve ser considerado, podendo ser mais elevado que o atraso de portas lógicas internas [3].

Sistemas digitais tradicionalmente são projetados no paradigma síncrono, isto é usam um sinal de relógio global para sincronizar as suas operações. Eles são bastantes populares devido à simplicidade de projeto, e disponibilidade de ferramentas comerciais CAD para síntese automática. Na tecnologia MOS-DSM, um sinal de relógio global requer atenção, devido à geração de ruído, emissão eletromagnética e consumo de potência. Além destes fatores, a distribuição do sinal de relógio é uma tarefa com complexidade crescente devido ao problema do *clock skew*, que leva a uma queda no desempenho. O *overhead* causado pelo sinal de relógio pode chegar a 130% em uma implementação VLSI (*Very Large Scale Integration*) [4] e agrava-se em plataformas empregando FPGAs, Fig. 1.

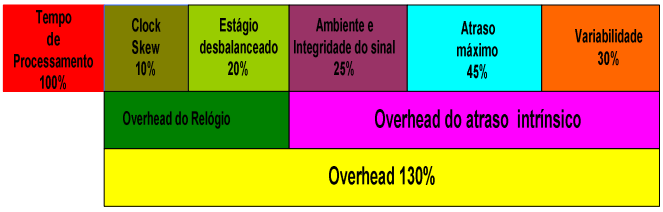

Fig. 1. *Overhead* do sinal de relógio.

Duarte L. Oliveira, duarte@ita.br, Tel +55-12-3947-6813, Fax +55-12-3947-6930, Kledermon Garcia, kedermonkg@iae.cta.br, Roberto d'Amore, damore@ita.br, Tel +55-12-3947-6876 - ITA - IEEA - Praça Marechal Eduardo Gomes, 50, Vila das Acácias - São José dos Campos - SP - Brasil.

Devido às características do paradigma assíncrono, no caso de controladores implementados em FPGAs comerciais, o estilo de projeto *micropipeline* tem mais facilidade de ser implementado nestas plataformas, devido à simplicidade do seu controle. Diferentes arquiteturas foram propostas para o estilo *micropipeline* linear [5-9]. Entretanto, grande parte destas arquiteturas é voltada para implementações VLSI, pois o controle é do tipo *full-custom* VLSI [6-8]. A arquitetura conhecida como MOUSETRAP, proposta em [9], tem um alto desempenho, é baseada em portas lógicas e pode ser implementada em FPGAs (ver Fig. 2).

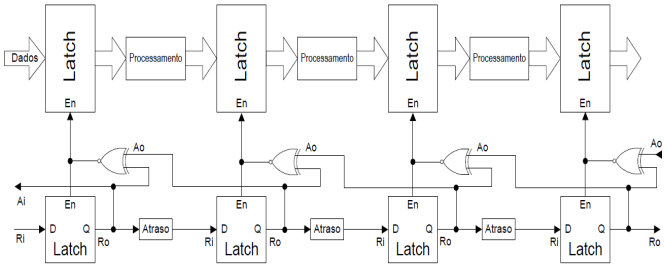

Fig. 2. *Micropipeline* linear MOUSETRAP de [9].

Neste artigo propomos uma arquitetura para o estilo de projeto *micropipeline* voltada para FPGAs (ver Fig. 3). A arquitetura emprega registradores do tipo *flip-flops*, devido a grande disponibilidade destes elementos em FPGAs. O controle é baseado em portas lógicas. Esta proposta é mais vantajosa que a arquitetura MOUSETRAP pois: a) obtêm tempo de latência menor e *throughput* maior; b) emprega um número menor de LUTs; c) emprega no controle duas funções Booleanas combinatórias soma-de-produto e tem o seu mapeamento facilitado pois não tem realimentação.

Este artigo está estruturado como segue: seção II mostra o projeto micropipeline em FPGAs; seção III apresenta o nosso controle pipeline; seção IV apresenta a nossa metodologia; seção V ilustra com um exemplo a nossa arquitetura; seção VI discute e compara arquiteturas micropipeline; seção VII finalmente as nossas conclusões e trabalho futuro.

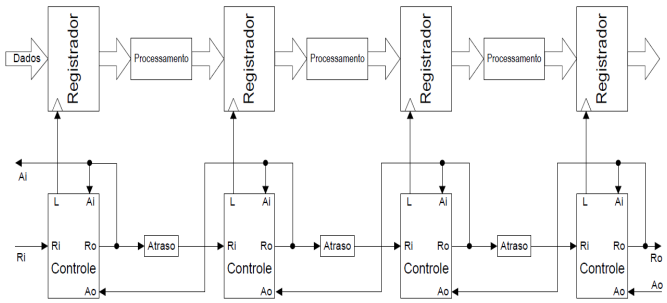

Fig. 3. Arquitetura proposta para *micropipeline* linear.

II. PROJETO MICROPIPELINE EM FPGAS

Sistemas digitais assíncronos operam por eventos e não possuem um sinal global para sincronizar as operações. A sincronização é realizada por protocolos do tipo *Handshake*. Um sistema digital assíncrono pode ser projetado em diferentes estilos: a) decomposição, controlador + *data-path* [10]; b) *micropipeline*, proposto por Sutherland (ver Fig. 4) [11]; c) composição com macromódulos [12,13]; d) dessincronização [14]. Os quatro estilos podem ser projetados em diferentes classes de circuitos assíncronos [15], onde a classe está relacionada com o modelo de atraso. Os principais desafios nos projetos de circuitos assíncronos são a maior complexidade no projeto e a falta de ferramentas para síntese automática. A complexidade é devida a condição do circuito, ser livre de risco (*hazard*) e de corrida crítica [15,16].

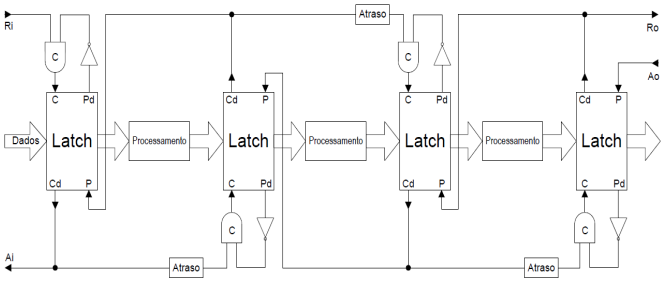

Fig. 4. *Micropipeline* linear de [11].

A abordagem síncrona é tradicionalmente empregada em projetos visando a implementação em *FPGAs* [17]. Esforços para prototipagem assíncrona foram relatados recentemente em FPGAs comerciais [18-20] e FPGAs assíncronas acadêmicas [21,22]. As dificuldades na implementação de sistemas assíncronos em FPGAs comerciais são:

a. Processo de mapeamento de funções Booleanas livre de risco em blocos lógicos (macrocélulas). As ferramentas comerciais de decomposição e mapeamento em LUTs de funções Booleanas não estão preparadas para satisfazer os requisitos de *hazard* lógico, podendo acarretar um funcionamento incorreto do circuito se uma intervenção manual para correção do problema não for realizada. A decomposição deve satisfazer os requisitos propostos em Sigel et al. [23]. O mapeamento deve satisfazer a suposição *isochronic fork* que é obtido pela escolha adequada das macro-células. [15,16].
b. Processo de roteamento interno entre as macrocélulas pode introduzir atrasos significativos. Estes atrasos podem resultar em *hazard* essencial e levar a um funcionamento incorreto do circuito [15,16]. O modelo de atraso do circuito define como solucionar o problema de *hazard* essencial: inserção de elementos de atraso nas linhas de realimentação ou empregar macrocélulas que satisfaçam a condição *isochronic fork*.

A arquitetura *micropipeline* pode ser linear ou não linear [24]. Neste trabalho apenas focamos o *micropipeline* linear. O estilo *micropipeline* tem como principal característica a simplificação do controle do pipeline, que é importante em sistemas assíncronos implementados em FPGAs. O controle é distribuído entre os estágios ou centralizado, sendo responsável por realizar a comunicação entre os estágios do *pipeline*. Esta comunicação emprega o protocolo do tipo *Handshake* entre o sinal Pedido (*Request*) e o sinal Aceite (*Acknowledge*) [15,16]. A comunicação entre os estágios pode ser realizada no protocolo de quatro fases (4-fases) ou no protocolo de duas fases (2-fases). Fig. 5 mostra, respectivamente, os comportamentos do protocolo de 4-fases e de 2-fases.

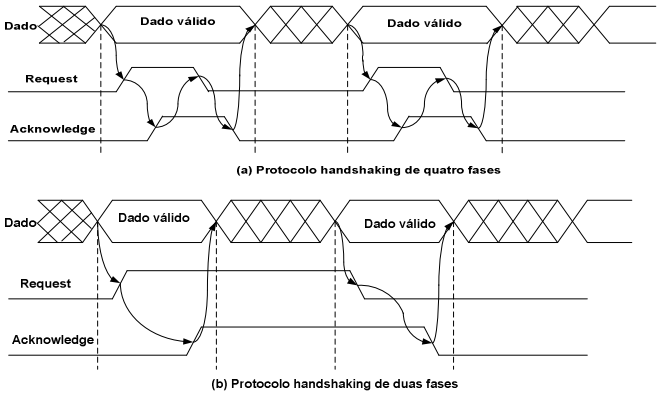

Fig. 5. Protocolo *handshake*: a) 4-Fase; b) 2-Fases.

O projeto *micropipeline* linear tem em duas variantes. Na primeira arquitetura, somente são empregados componentes do paradigma síncrono (*single-rail*) e elementos de atraso entre estágios. O cálculo do elemento de atraso é realizado através do caminho crítico do estágio. Neste trabalho empregamos essa arquitetura, também chamada de "*micropipeline bundled-data*" (ver Fig. 3). A implementação *bundled-data* é um dos esquemas de codificação dos dados usados em circuitos assíncronos. Ele representa N bits de dados com N+2 linhas chamada "*bundled*", onde as duas linhas adicionais são os sinais *handshake request* e *acknowledge*.

A segunda arquitetura emprega componentes *dual-rail* (ver Fig. 6) [15,16]. Os componentes são sintetizados usando códigos *Delay Insensitive* (DI) ou *dual-rail*. No código *dual-rail* cada sinal é codificado com dois bits. Para o sinal *a*, temos *a0a1=00* (nulo), *a0a1=01* (1), *a0a1=10* (0) e *a0a1=11* (nunca ocorre). O projeto empregando este código gera um sinal de término da operação sem a necessidade do elemento de atraso e com um circuito relativamente simples. Fig. 7 mostra uma porta AND *dual-rail* projetada a partir de portas AND *single-rail* e a porta OR *single-rail*. Para deixar a porta AND *dual-rail* robusta, as portas AND *single-rail* devem ser substituídas por *latches* C. Fig. 8 mostra um somador de dois bits *dual-rail* com término de operação, usando *latches* C. A arquitetura *micropipeline* linear *dual-rail* pode ser implementada na plataforma FPGA, mas há um aumento significativo de macrocélulas que pode comprometer o desempenho e consumo de potência. As vantagens são o desaparecimento de todos os elementos de atraso, o aumento de robustez e o aumento de velocidade.

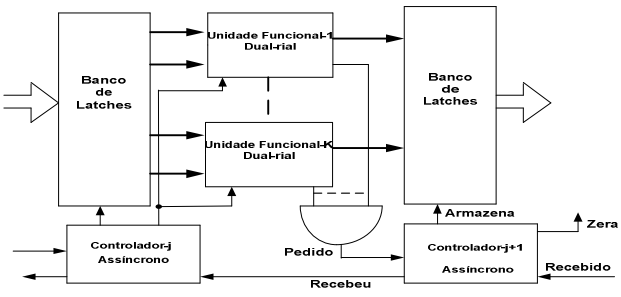

Fig. 6. *Micropipeline* com unidades funcionais *dual-rail*.

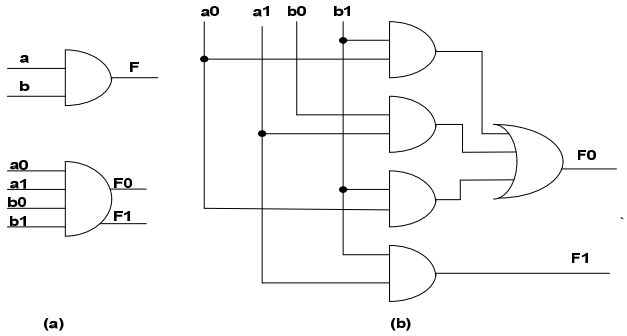

Fig. 7. Porta AND de duas entradas *dual-rail*: a) símbolo; b) projeto usando portas básicas (*single-rail*).

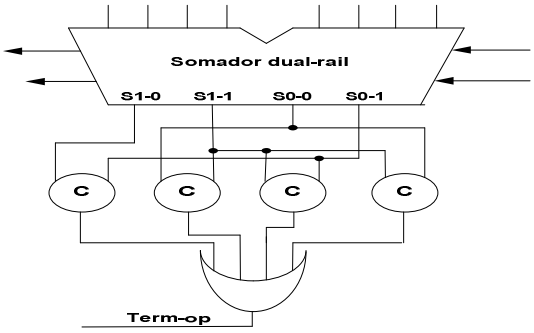

Fig. 8. Somador de 2 bits *dual-rail* com termino de operação.

III. CONTROLE *PIPELINE* PARA FPGA

A arquitetura *micropipeline* proposta opera no protocolo *handshake* de 2-fases e, portanto, o pedido (*request*) de processamento ocorre nas duas bordas deste sinal. Fig. 9 mostra o esquema geral de um estágio, que é composto por um registrador baseado em *flip-flops* e controle. O controle foi especificado no modo rajada [25] (ver Fig. 10). Ele é composto pelos sinais: *Ri* (*request input*), *Ao* (*acknowledge output*), *Ai* (*acknowledge input*), *L* (*load*) e *Ro* (*request output* – pedido-armazenar). Fig. 11 mostra o circuito lógico do controle.

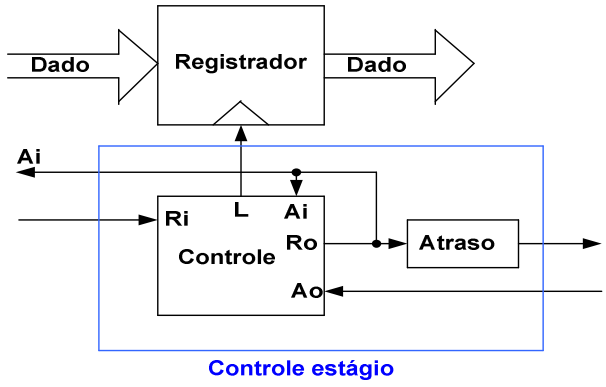

Fig. 9. Esquema do *micropipeline* linear voltado para FPGA.

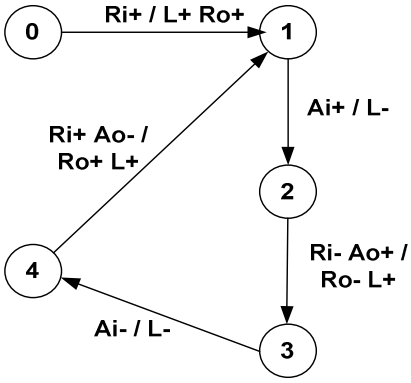

Fig. 10. Especificação modo rajada do controle proposto.

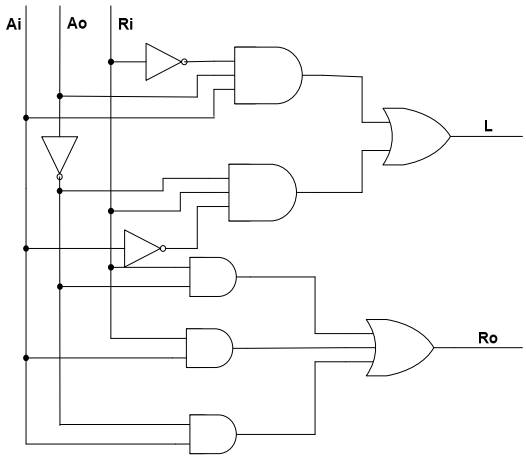

Fig. 11. Controle: circuito lógico.

IV. SÍNTESE DE SISTEMAS *PIPELINE*

O procedimento adotado para sintetizar sistemas *micropipeline bundled-data* procura empregar componentes e ferramentas de síntese do paradigma síncrono (ver Fig. 12). A síntese comportamental parte do grafo de fluxo de dados (GFD) que representa operações e a sua dependência de dados (semelhante ao paradigma síncrono). O GFD é usado como entrada para o método de síntese proposto. Usando os algoritmos de escalonamento de operações (*scheduling*) e de alocação de recursos, as operações e recursos são assinalados em cada ciclo (*time slots*) do GFD escalonado [26]. O método parte da descrição em GFD do algoritmo a ser sintetizado, e pode ser dividido três passos:

1) A partir do GFD escalonado, gerar o *data-path pipeline* usando componentes *single-rail* e a síntese *pipeline* conforme [26].
2) Para cada estágio do *data-path pipeline*, identificar o caminho crítico e calcular o respectivo elemento de atraso (elemento de atraso da Fig. 14).
3) Usando os resultados dos passos (1) e (2) realizar o mapeamento para a arquitetura *micropipeline* linear proposta.

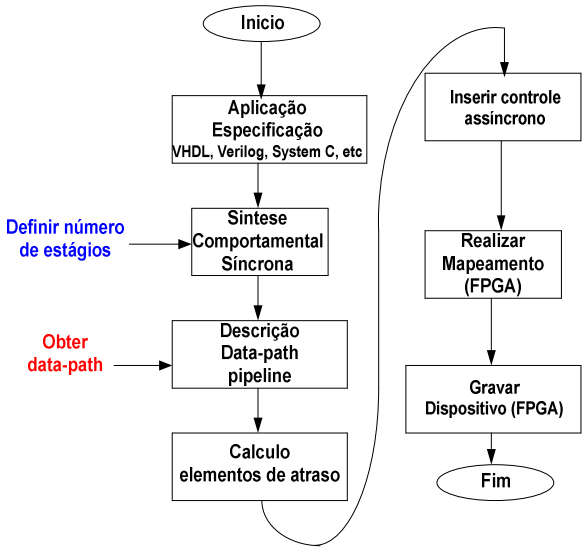

Fig. 12. Fluxo de projeto: *micropipeline* voltado para FPGA.

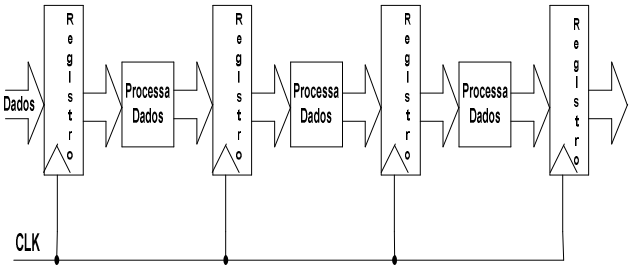

Fig. 13. Arquitetura *pipeline* linear síncrona.

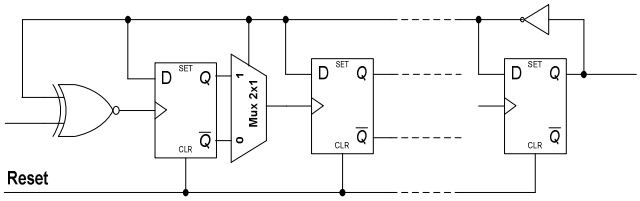

Fig. 14. Elemento de atraso simétrico baseado em *flip-flops* voltado para FPGAs.

V. APLICAÇÃO

Para ilustrar a nossa arquitetura *micropipeline*, usamos o projeto do filtro FIR de ordem cinco em uma versão assíncrona:

$$Y[t] = \sum_{i=0}^{N-1} H[i] \times X[t-1] \qquad (1)$$

Fig. 15 mostra o GFD escalonado do filtro FIR obtido pelo algoritmo *List scheduling* com as seguintes restrições de recursos: dois multiplicadores e um somador. O passo 1 do nosso método gera o data-path *pipeline* de seis estágios, que foi obtido pelo procedimento da síntese comportamental de [26] (ver Fig. 16). Fig. 17 mostra o filtro FIR de quinta ordem implementado conforme a arquitetura proposta, *pipeline* de seis estágios (passos *2* e *3*).

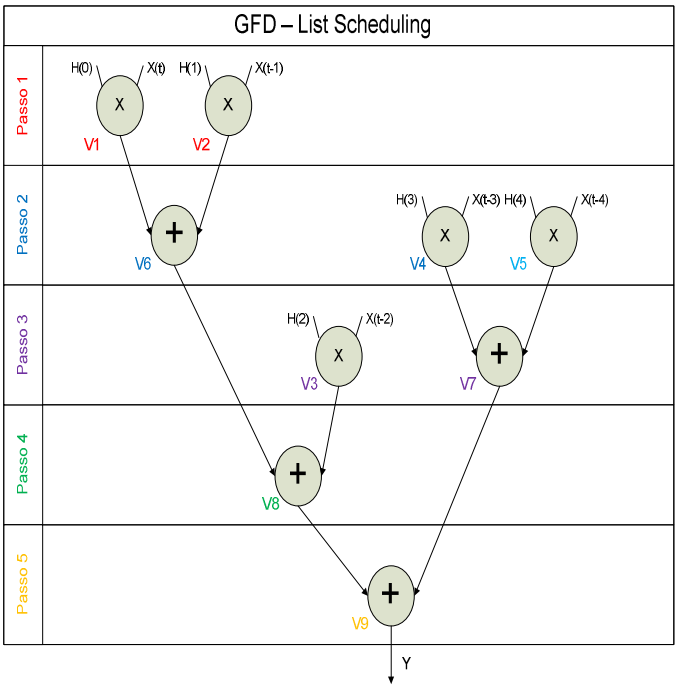

Fig. 15. Filtro FIR: GFD escalonado.

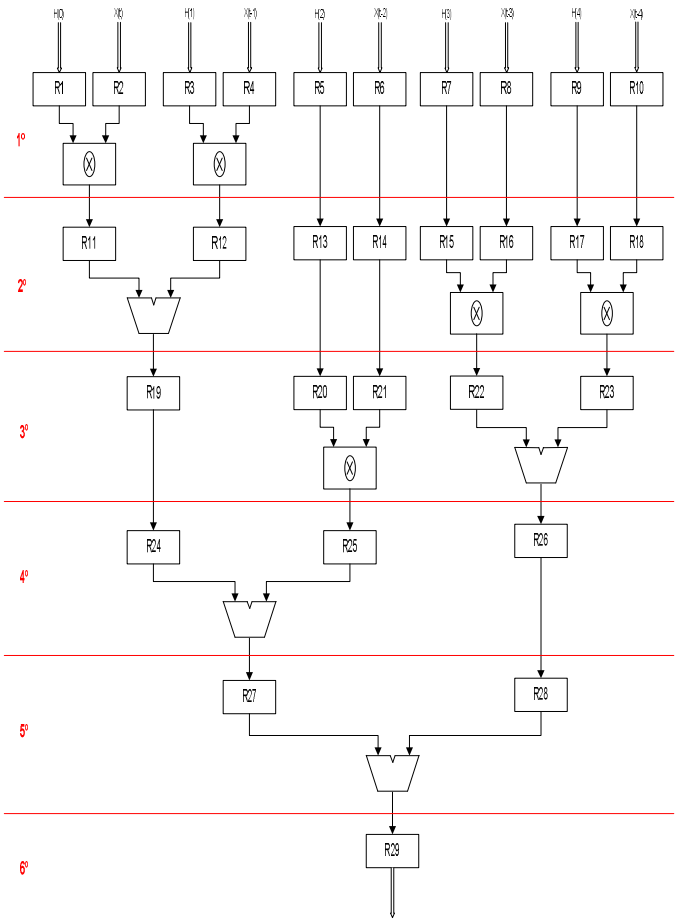

Fig. 16. Filtro FIR: *data-path - pipeline* síncrono.

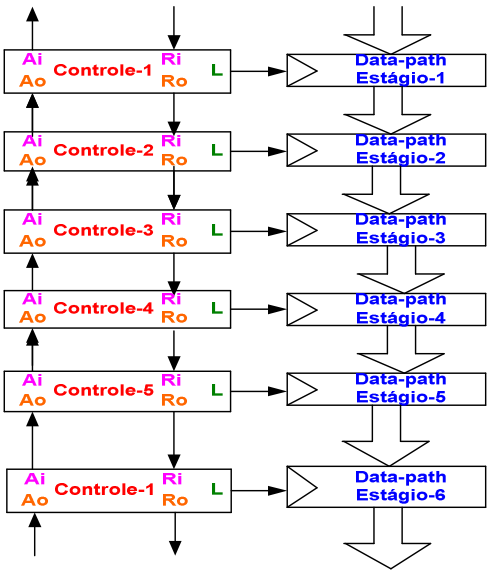

Fig. 17. Filtro FIR implementado por *micropipeline*.

VI. DISCUSSÃO & RESULTADOS

A fim de demonstrar o desempenho da arquitetura *pipeline* proposta, foi realizada uma comparação com a arquitetura *pipeline* MOUSETRAP. As simulações foram realizadas no software QUARTUS II versão 9.1 da ALTERA [27], família STRATIX II no dispositivo EP2S15F484C3. Fig. 18 e 19 mostram respectivamente as simulações do MOUSETRAP do "controle isolado" e do "controle total" (seis controles) do filtro FIR.

Fig. 20 e 21 mostram, respectivamente, as simulações do *pipeline* proposto nas condições: "controle isolado" e do "controle total". O tempo de latência (para armazenamento) no controle proposto foi reduzindo em 18%. A penalidade no tempo de latência para o sinal *Ro* (*request* de saída) é de 4% no controle proposto. Na área houve uma redução de 33% no controle proposto, respectivamente 3 e 2 LUTs.

Fig. 22 e 23 mostram, respectivamente, as simulações do filtro FIR de seis estágios nas arquiteturas MOUSETRAP e proposta. A Tabela I apresenta resultados do filtro FIR nas arquiteturas MOUSETRAP e proposta, que envolve tempo de latência, *throughput* (MOPS – 10^6 operações por segundo), área e potência dissipada.

Os resultados mostram que a arquitetura proposta obteve uma redução de 24% no tempo de latência, um aumento no *throughput* de 3%, uma redução de 3% na potência dissipada e uma redução de 63% nas macrocélulas usadas.

TABELA I RESULTADOS: ARQUITETURAS

Arquitetura Micropipeline	Tempo de Latencia	Throughput MOPS	Número de LUTs	Potência dissipada
MOUSETRAP	31,028ns	141,2	391	753,23mw
Proposta	**23,636ns**	**146,3**	**219**	**734,34mw**

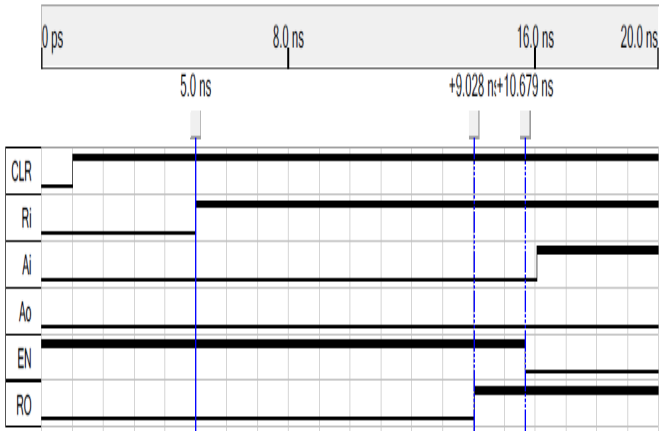

Fig. 18. MOUSETRAP - "controle isolado"

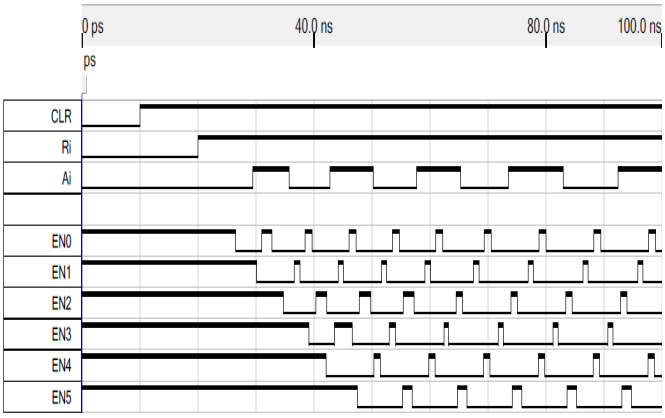

Fig. 19. MOUSETRAP - "controle total"

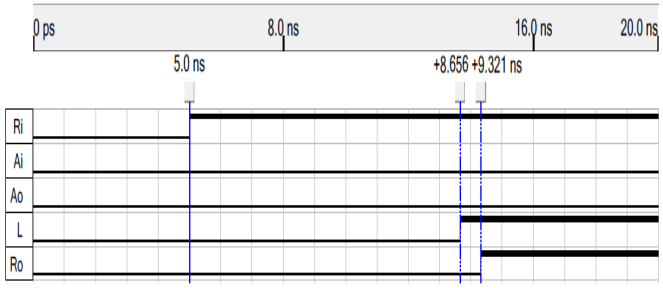

Fig. 20. Arquitetura proposta: "controle isolado".

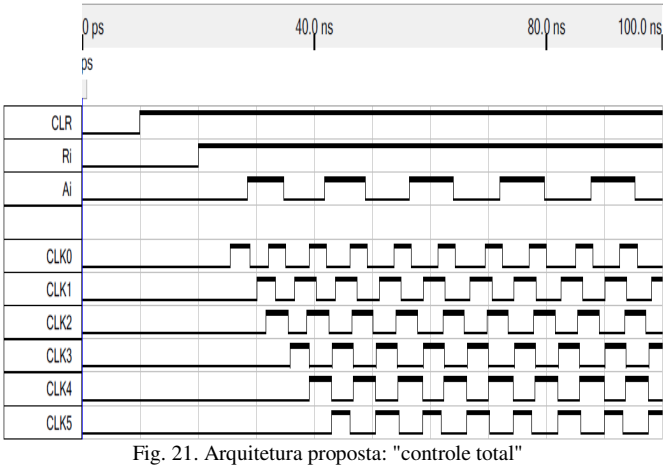

Fig. 21. Arquitetura proposta: "controle total"

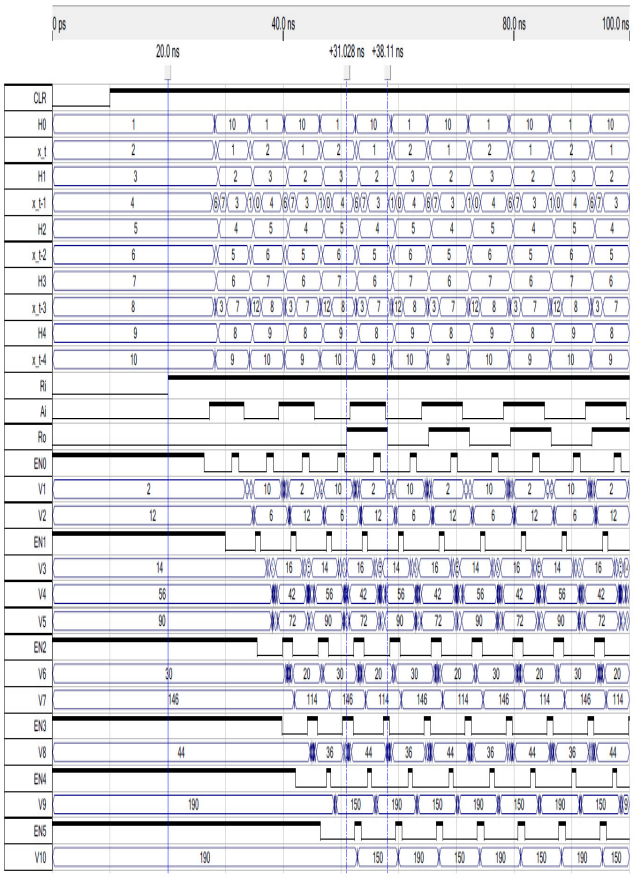

Fig. 22. Simulação:Filtro FIR *micropipeline* MOUSETRAP.

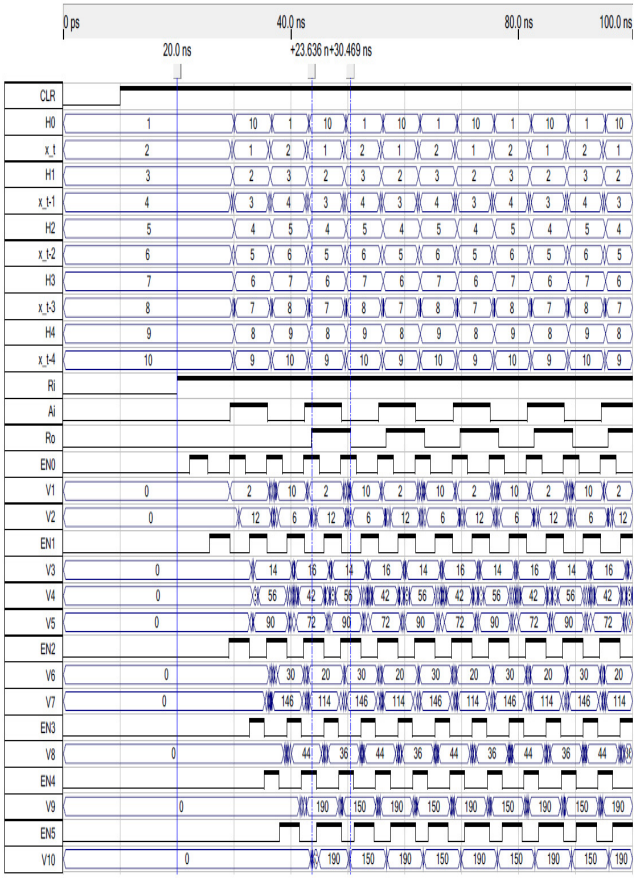

Fig. 23. Simulação: Filtro FIR *micropipeline* proposto.

VII. CONCLUSÕES

O estilo *micropipeline* linear do paradigma assíncrono é vantajoso em aplicações com arquitetura *pipeline* porque simplifica o controlador. Neste artigo nós propusemos uma nova arquitetura *micropipeline* voltada para implementações em FPGAs. O controle é um circuito combinatório, que acarreta simplificação e redução de área. Os registradores usados são baseados em *flip-flops*, que permitem uma redução de recursos do FPGA empregando os *flip-flops* disponíveis nestes dispositivos.

Através de um caso de estudo mostramos que a nossa arquitetura tem um desempenho melhor que a arquitetura MOUSETRAP.

REFERÊNCIAS

[1] C. Constantinescu, "Trends and Challenges in VLSI Circuits Reliability," *IEEE Micro*, 23 (4), 2003.

[2] K. D. Muller-Glaser, et. al. "Multiparadigm Modeling in Embedded Systems Design", *IEEE Trans. on Control Systems Technology*, vol. 12, no. 2, March 2004.

[3] D. Goldhaber-Gordon, et al., "Overview of Nanoelectronic Devices," *Proc. of the IEEE*, vol. 85, No. 4, pp.521-540, April 1997.

[4] J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou, "Coping with the variability of combinational logic delays," *ICCD*, pages 505–508, 2004.

[5] S. B. Furber and P. Day, "Four-Phase Micropipeline Latch Control Circuits," *IEEE Trans. on VLSI Systems*, vol.4, no. 2, pp.247-253, June, 1996.

[6] G. S. Taylor and G. M. Blair, "Reduced Complexity Two-Phase Micropipeline Latch Controller," *IEEE Journal of Solid-State Circuits*, vol. 33, Nro. 10, pp.1590-1593, October, 1998..

[7] M. Singh and S. M. Nowick, "The Design of High-Performance Dynamic Asynchronous Pipelines: Lookahead Style," *IEEE Trans. on VLSI Systems*, vol.15, no. 11, pp.1256-1269, November, 2007.

[8] M. Singh and S. M. Nowick, "The Design of High-Performance Dynamic Asynchronous Pipelines High-Capacity Style", *IEEE Trans. on VLSI Systems*, vol.15, no.11, ´pp.1270-1283,November, 2007.

[9] M. Singh and S. M. Nowick, "MOUSETRAP: High-Speed Transition-Signaling Asynchronous Pipelines", *IEEE Trans. on VLSI Systems*, vol.15, no. 6, pp.684-698, June, 2007.

[10] T. Konishi, N. Hamada and H. Saito, "A Control Circuit Synthesis Method for Asynchronous Circuits in Bundled-Data Implementation", Seventh International Conference on Computer and Information Technology, pp.847-852, 2007.

[11] I. E. Sutherland, "Micropipelines", *Communication of the ACM*, vol. 32, No.6, pp.720-738, June, 1989.

[12] S. H. Ungle, "A Building Block Approach to Unclocked Sytems", IEEE ICCD, pp. 339-348. 1993.

[13] S. F. Nielsen, et. al., "Behavioral Synthesis of Asynchronous Circuits Using Syntax Directed Translation as Backend", *IEEE Trans. on VLSI Systems,* vol. 17, no. 2, pp. 248-261, February, 2009.

[14] J. Cortadella, et al., "Desynchronization: Synthesis of Asynchronous Circuits from Synchronous specifications," *IEEE Trans. on CAD of Int. Circuits and Systems*, vol. 25, Nro. 10, pp. 1904-1921, October 2006.

[15] C. J., Myers, "*Asynchronous Circuit Design*", Wiley & Sons, Inc., 2004, 2ª edition.

[16] J. Sparsø e S. Furber, "*Principles of Asynchronous Circuits Design*", Kluwer Academic Publishers, 2001.

[17] J. J. Rodriguez, et. Al., "Features, Design Tools, and Applications Domains of FPGAs", *IEEE Trans. on Industrial Electronics*, vol. 54, No. 4, pp.1810-1823, August 2007.

[18] E. Brunvand, "Using FPGAs to Implement Self-Timed Systems", *Journal of VLSI Signal Processing*, Special issue on field programmable logic, vol.6(2), pp.173-190, August, 1993.

[19] M. Tranchero and L. M. Ryneri, "Implementation of Self-Timed Circuits onto FPGAs Using Commercial Tools", 11th Euromicro Conf. on Digital System Design Architectures, Methods and Tools, pp.373-380, 2008.

[20] T.N. Prabakar G. Lakshminarayanan, K. K. Anilkumar, "FPGA Based Asynchronous Pipelined Multiplier with Intelligent Delay Controller," International SOC Design Conference, 304-309, 2008.

[21] R. Payne, “Asynchronous FPGA architectures,” *IEE Proc. Comp. Digit. Tech.*, vol.143, no.5, September, pp.282-286, 1996.
[22] N. Huot, et. al., “FPGA architecture for multi-style asynchronous logic,” Proc. Of the Design, Automation and Test in Europe Conference and Exhibition, 2005.
[23] P. Sigel, G. De Michele and D. Dill, “Decomposition methods for library binding of speed-independent,” Proc. Int. Conf. Computer-Aided Design, pp.558-565, 1994.
[24] R. O. Ozdag M. Singh, P. A. Beerel S. M. Nowick, “High-Speed Non-Linear Asynchronous Pipelines,” Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exhibition (DATE.02), 2002.
[25] S. M. Nowick, “*Automatic Synthesis of Burst-Mode Asynchronous Controller*”, PhD thesis, Stanford University, 1993.
[26] D. D. Gajski, “*Principles of Digital Design*,” Prentice Hall, 1997.
[27] Altera Corporation, 20013, www.altera.com.
.