Sistemas Assíncronos em FPGAs: Uma Alternativa para Projeto Digital Embarcado

Duarte L. Oliveira
Divisão de Engenharia Eletrônica do Instituto Tecnológico de Aeronáutica – ITA – IEEA
Marechal Eduardo Gomes, 50 – CEP 12.228-900 – SJC – SP – Brazil
email: duarte@ita.br

***Resumo* — Projeto digital embarcado possui requisitos críticos, tais como consumo de potência, robustez, velocidade, etc. O paradigma assíncrono possui características interessantes que pode ser uma alternativa de projeto. Neste artigo discutimos os estilos do projeto assíncrono voltado para os sistemas embarcados. Mostramos a viabilidade de implementar circuitos assíncronos em dispositivos FPGAs que são uma alternativa de projeto rápido e de custo reduzido. Também propomos duas arquiteturas assíncronas voltadas para FPGAs do estilo *micropipeline*. Uma arquitetura é robusta e a outra uma versão da arquitetura MOUSETRAP. Mostramos as duas arquiteturas em uma aplicação conhecida que é um filtro FIR de terceira ordem.**

***Palavras chaves* — Lógica assíncrona, micropipeline, data-path assíncrono, controlador, FPGA.**

I. INTRODUÇÃO

Sistemas digitais no ambiente embarcado requerem alta capacidade de integração, alta velocidade e baixo consumo de energia [1]. Eles podem necessitar de baixa interferência eletromagnética, serem robustos aos efeitos radiativos, como também a variações de temperatura e tensão de alimentação. Um caminho que pode satisfazer os requisitos do ambiente embarcado é a implementação destes sistemas na tecnologia FPGA (*Field Programmable Gate Array*) [2,3]. Os dispositivos FPGAs tornaram-se um meio popular de implementar circuitos digitais. Tecnologia FPGA tem crescido consideravelmente nos últimos anos, gerando FPGAs de até 50 milhões de portas, permitindo assim que sistemas digitais complexos possam ser programados em tais dispositivos [3,4]. FPGAs de alto desempenho são implementados na tecnologia MOS *Deep-Sub-Micron* (MOS-DSM). Essa tecnologia necessita operar com baixo ruído e a diferença entre o atraso máximo e mínimo nas linhas e portas é maior quando comparado com outras tecnologias MOS, e o atraso em uma linha pode ser maior que o atraso em uma porta [5]. Sistemas digitais síncronos usam um sinal de relógio global para sincronizar as suas operações e são bastantes populares devido à simplicidade de projeto. Também há uma oferta abundante de ferramentas CAD comerciais para síntese automática. Um sério problema na tecnologia MOS-DSM é conviver com o sinal de relógio global, porque ele é um grande causador de ruído, de alta emissão eletromagnética, consome uma parte significativa da potência e definir a distribuição do sinal de relógio é uma tarefa com complexidade crescente (por exemplo: relógio defasado – *clock skew*). Análise de temporização de circuitos digitais MOS-DSM síncronos de alta integração é extremamente difícil. Uma característica comum nos sistemas eletrônicos embarcados é o fato de serem alimentados por bateria. Como são alimentados por bateria é desejável que as baterias tenham uma longa vida útil, portanto a potência dissipada é um parâmetro muito importante na concepção de tais sistemas. Em um sistema digital á parte seqüencial é o principal contribuinte para a dissipação de potência dinâmica [6]. Estudos recentes têm mostrado que em tais sistemas o relógio consome uma grande percentagem (15% a 45%) da potência do sistema [6]. Uma interessante alternativa para projeto digital embarcado, porque ele elimina os problemas causados pelo sinal de relógio e aumenta a robustez é o paradigma assíncrono.

Sistemas digitais assíncronos operam por eventos não possuem um sinal global que sincroniza as operações. A sincronização é realizada por protocolos do tipo *Handshaking*. Sistema digital assíncrono pode ser projetado em três diferentes estilos: **a)** decomposição controlador + *data-path*; **b)** *micropipeline;* **c)** composição com macromódulos. Os três estilos podem ser projetados em diferentes classes de circuitos assíncronos. A classe define em que modelo de atraso o circuito opera corretamente e em que modo de operação o circuito se comunica com o ambiente [7]. Inicialmente os circuitos assíncronos podem ser classificados em duas classes: **a)** portas e linhas com atrasos delimitados (*bounded gate and wire delay*); **b)** portas e linhas com atrasos quaisquer (indefinidos), mas finitos (*unbounded gate and wire delay*). Dois importantes circuitos assíncronos que obedecem ao modelo de atraso (*a)* são: modo-rajada e extensões (*burst-mode circuits*) [8,9,10] e temporizado (*timed circuit*) [11]. O circuito insensível ao atraso (*delay insensitive circuit – DI*) obedece ao modelo de atraso (*b*) [12]. Este modelo é o mais robusto e isento de qualquer análise de temporização. Martin [13] mostra que a aplicação deste modelo é muito restrita. Duas variantes menos restrita deste modelo de atraso são: **a)** circuitos independentes da velocidade (*speed independent circuit – SI*). Eles obedecem ao modelo onde o atraso nas portas é indefinido, mas finito e nas linhas o atraso é zero [14]; **b)** circuitos quase insensíveis ao atraso (*quasi delay insensitive – QDI*). Eles obedecem ao modelo (*b)*, mas com a restrição *isochronic fork*. Esta restrição diz que linhas com *fan-out >1* (garfo) os atrasos são iguais [13,15]. A comunicação do circuito com o ambiente ou é realizada no modo fundamental (MF) (por exemplo: modo-rajada) [7] ou no modo Entrada/Saída (M_E/S) (por exemplo: DI, SI, QDI) [7,15]. No MF a mudança de um novo conjunto de entradas o circuito deve estar estabilizado. No M_E/S a mudança de um sinal de saída pode imediatamente habilitar a mudança de um sinal na entrada.

Circuitos assíncronos não apresentam problemas relacionados com a distribuição do relógio (por exemplo relógio defasado). Os circuitos resultantes têm baixa interferência eletromagnética e baixo consumo de potência [15]. A classe de circuitos assíncronos que mais satisfaz os requisitos do **projeto digital embarcado é a classe QDI**. Esta classe possui importantes propriedades que são muito interessantes para aplicação embarcada, tais como:

a. Potencial de melhor tempo de latência;
b. Robustos a variações de temperatura e tensão de alimentação;
c. Robusto a radiação do tipo SEU (*Single-Event Up-set*) [16] e falhas do tipo atraso e do tipo *Stuck-at* (classes de falhas facilmente testadas);
d. Altamente modular permitindo grande reusabilidade e que são facilmente usados como propriedade intelectual (*intellectual property – IP*) [17];
e. Melhor desempenho no projeto de sistemas de segurança (por exemplo, criptografia) [18,19].

Dois são os pontos fracos dos circuitos assíncronos: dificuldade no projeto e falta de ferramentas para síntese automática. A dificuldade no projeto é que o circuito deve ser livre de risco (*hazard*) e de corrida crítica [7,15].

Projeto digital embarcado em FPGAs [20]. Estes dispositivos são bastantes populares para prototipagem e produção de circuitos digitais, isto é devido ao seu baixo custo e tempo curto de projeto. O seu foco tem sido circuitos digitais síncronos. Houve alguns recentes esforços para prototipagem assíncrona. Ela foi realizada em FPGAs comerciais [21,22] e FPGAs assíncronas acadêmicas [23,24]. Existem duas razões porque FPGAs comerciais têm dificuldades em implementar controladores QDI:

a. *Processo de mapeamento* de funções *Booleanas* livre de risco em blocos lógicos (macro células) pode introduzir risco (*hazard*). As ferramentas comerciais de decomposição e mapeamento em FPGAs baseadas em *look-up tables* (LUTs) de funções *Booleanas* livre de risco não estão preparadas para satisfazer o modelo QDI, portanto é necessária uma intervenção manual. A decomposição deve satisfazer os requisitos propostos por Sigel et al. [25]. O mapeamento deve satisfazer a suposição *isochronic fork*.
b. *Processo de roteamento* interno entre blocos lógicos (macro-células) pode introduzir atrasos significativos que pode resultar em risco essencial. O modelo QDI é livre de risco essencial porque as linhas com fan-out >1 devem satisfazer a suposição *isochronic fork,* e os atrasos nas portas inversoras são desprezíveis porque as LUTs são portas complexas baseadas em mux´s. Cada bloco lógico em FPGAs baseadas em LUTs satisfaz a suposição *isochronic fork*, mas as conexões entre os blocos lógicos não satisfazem esta suposição. Uma intervenção manual deve escolher os blocos lógicos que satisfaçam a suposição *isochronic fork*.

Neste artigo apresentamos uma revisão dos estilos da síntese assíncrona, como também uma análise das dificuldades do projeto assíncrono em FPGAs quando aplicado em ambientes embarcados. Também propomos duas arquiteturas *micropipeline* voltadas para FPGA. Uma é arquitetura QDI e a outra é uma variante da arquitetura MOUSETRAP [26].

Este artigo está estruturado como segue: seção II apresenta os três estilos do projeto assíncrono; seção III apresenta as duas arquiteturas; seção IV ilustra com um exemplo o projeto assíncrono; seção V discute cada estilo do projeto assíncrono; seção VI finalmente as nossas conclusões e trabalhos futuros.

II. MÉTODOS DE SÍNTESE

Nesta seção mostramos os procedimentos de projeto para os estilos decomposição, *micropipeline* e composição.

A. Decomposição

Este estilo segue o procedimento tradicional RTL (*register transfer level*) do projeto síncrono, que descreve o projeto no grafo de fluxo de dados e controle (GFDC) [27]. Figura 1 mostra a arquitetura baseada em decomposição. O modulo *data-path* está relacionado com o processamento de dados e o modulo controlador está relacionado com a seqüência da execução das operações. Na etapa relacionada com a síntese comportamental, ferramentas de decomposição funcional, escalonamento e alocação do mundo síncrono podem ser usadas na síntese do data-path otimizado [28,29] (ver Fig. 2). No projeto *data-path* há duas variantes que podem ser usadas na implementação em FPGAs:

a. *data-path síncrono*. Neste projeto as unidades funcionais (UF) são sintetizadas na forma convencional (*single-rail*). O controlador assíncrono tem uma variável que inicia as operações na transição de estado (*Inic-tran*) e outra variável que sinaliza o fim das operações na transição de estado (*Fim-tran*). Elemento de atraso é inserido para gerar o inicio e termino da transição de estado. Ele é calculado nos tempos de propagação do caminho crítico do *data-path*. Figura 3 mostra a arquitetura assíncrona com *data-path* síncrono.
b. *data-path assíncrono*. Neste projeto as unidades funcionais são sintetizadas usando códigos DI [15], por exemplo o código *dual-rail*. No código *dual-rail* cada variável é codificada com dois bits. Para a variável ***a***, temos *a0a1=00* (nulo), *a0a1=01* (1), *a0a1=10* (0) e *a0a1=11* (nunca ocorre). Esta variante gera o sinal de termino da operação sem a necessidade do elemento de atraso e com um circuito relativamente simples. Há duas formas para projetar unidades funcionais *dual-rail*: **1)** projetar a UF na forma convencional e posteriormente substituir as portas *single-rail* por portas *dual-rail* (conversão); **2)** projetar a UF como *dual-rail* (síntese direta). Figuras 4a,b mostram o projeto da porta AND de duas entradas *dual-rail*. Este método usa portas convencionais. Figura 5 mostra a função soma (S=AB´C´ + A´BC´+ A´B´C + ABC) de 1 bit projetada como *dual-rail* por conversão. Figura 6 mostra a UF somadora de 1 bit *dual-rail* projetada por síntese direta. O circuito de detecção de realização da operação é obtido usando latch C. Figuras 7a,b,c mostram o latch C *single-rail* e a Fig. 8 mostra a UF somadora com o circuito de detecção de término de operação.

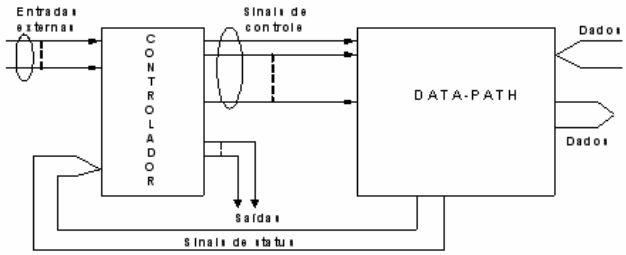

Fig. 1. Arquitetura: decomposição.

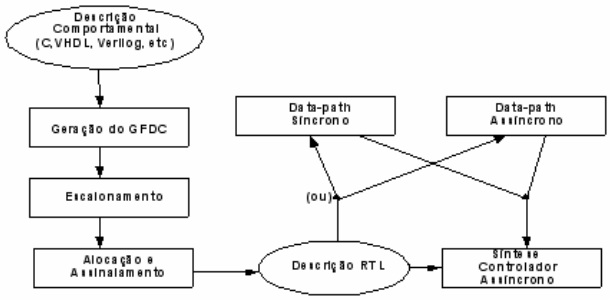

Fig. 2. Fluxo de projeto: decomposição.

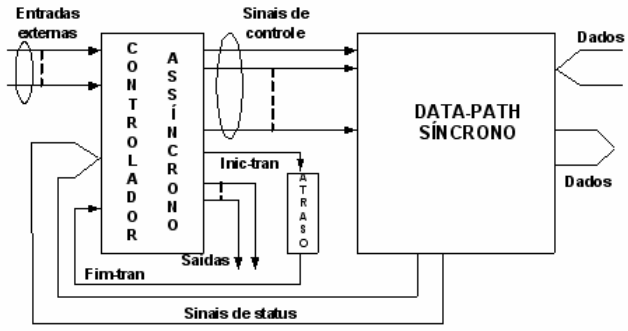

Fig. 3. Arquitetura assíncrona com *data-path* síncrono.

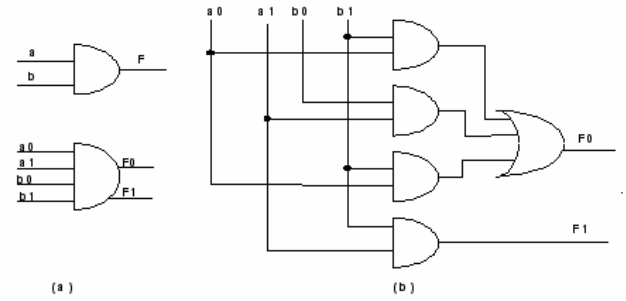

Fig. 4. Porta AND de duas entradas *dual-rail*: a) símbolo; b) projeto usando portas básicas (*single-rail*).

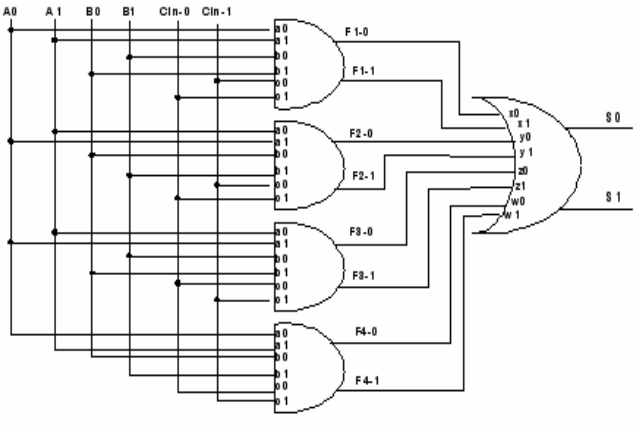

Fig. 5. Conversão: parte do somador completo de 1 bit *dual-rail*.

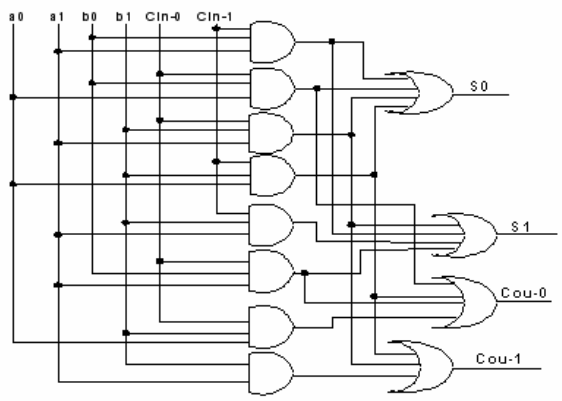

Fig. 6. Unidade funcional *dual-rail*: somador completo de 1 bit.

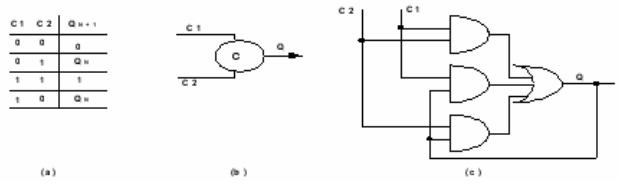

Fig. 7. *Latch* C: a) tabela de operação; b) símbolo; c) projeto com portas básicas.

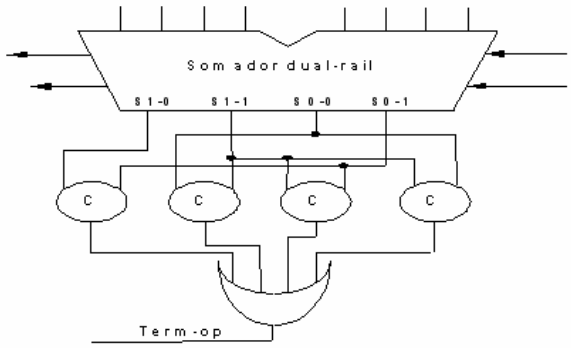

Fig. 8. Somador de 2 bits *dual-rail* com termino de operação.

A implementação de controladores assíncronos em FPGAs deve satisfazer os requisitos livre de risco. Estes requisitos que foram citados na seção I devem ser preservados na compilação. Projeto do controlador assíncrono pode ser realizado por de-sincronização, mapeamento direto e síntese lógica:

a) Na *de-sincronização,* diferentes estratégias foram propostas, por exemplo [30]. Na essência o projeto do controlador é realizado na forma síncrona através do uso de ferramentas comerciais e posteriormente é transformado em assíncrono.

b) Na *síntese lógica* o procedimento segue as etapas de assinalamento de estados e minimização lógica; os dois principais métodos são: **1)** máquinas de estado finito do tipo modo rajada. Elas operam no modo fundamental e aceitam a especificação modo-rajada e extensões [8,9,10]; **2)** máquinas baseadas em grafos. Elas operam no modo M_E/S e aceitam a especificação grafo de transição de sinais (GTS) que é baseada em Petri-net [14].

c) No *mapeamento direto* o circuito é extraído diretamente da especificação. Um método interessante foi proposto por Sokolov [31] que usa a especificação GTS e os circuitos resultantes são QDI.

B. Micropipeline

Este estilo de síntese denominada *micropipeline* foi proposto por Sutherland [32]. Este estilo tem como principal característica á simplificação da parte do controle, e ele pode ser decomposto em duas variantes:

a) usa unidades funcionais *single-rail* e elementos de atraso entre estágios. O calculo do elemento de atraso é realizado através do caminho crítico do estágio. Diferentes propostas foram feitas para esta variante. Destacamos as arquiteturas MOUSETRAP e HC [26,33]. Figura 9 mostra uma arquitetura *pipeline* geral com unidades funcionais *single-rail.* O controlador do *micropipeline* sem o sinal de *reset* foi especificado usando o grafo multi-rajada (GMR) [10]. Este *micropipeline* processa no protocolo *handshaking* de duas fases [7,15]. Figura 10 mostra a especificação GMR deste controlador.

b) usa unidades funcionais *dual-rail*. Figura 11 mostra a arquitetura geral com unidades funcionais *dual-rail*. Diversas propostas foram feitas para esta variante, por exemplo [34,35]. Figura 12 mostra a especificação multi-rajada do controlador. Este *micropipeline* processa no protocolo *handshaking* de quatro fases [7,15].

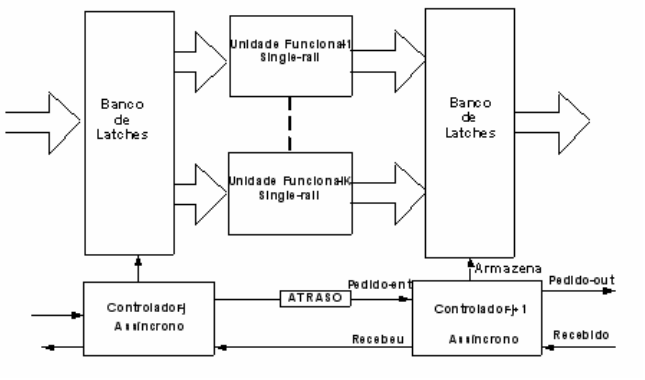

Fig. 9. Micropipeline com unidades funcionais single-rail.

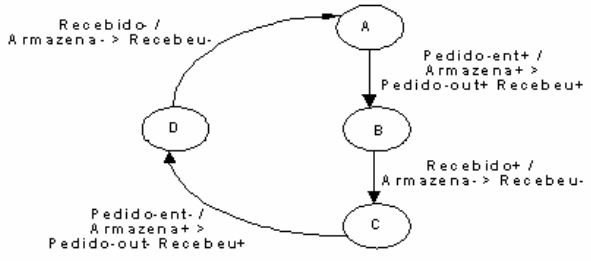

Fig. 10. Especificação do controlador do Mic-SR: Grafo Multi-rajada.

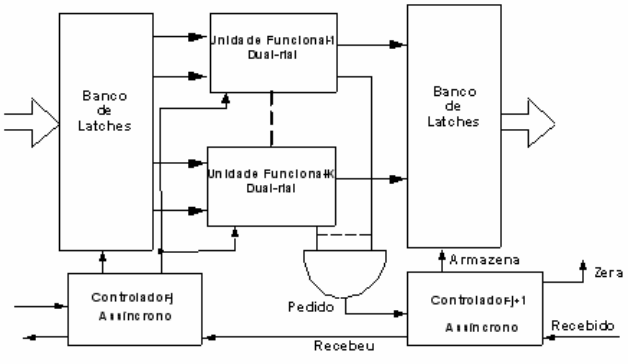

Fig. 11. Micropipeline com unidades funcionais dual-rail.

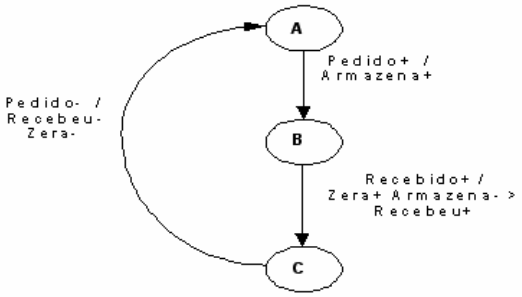

Fig. 12. Especificação do controlador Mic_DR: Multi-rajada.

C. Composição de macromódulos

Este estilo inicialmente proposto por Martin [12] tem como vantagem a possibilidade da síntese direta. Diversas ferramentas foram propostas para síntese direta. Elas partem das mais variadas linguagens [15], tais como: OCCAM, CSP, TANGRAN, Balsa, etc. Estas ferramentas possuem uma biblioteca de macromódulos, onde o algoritmo a ser sintetizado é mapeado nesses módulos. Figura 12 mostra o projeto de um multiplicador baseado em somas e deslocamentos que foi sintetizado com macromódulos propostos por Ungle [36]. Estes macromódulos são máquinas de estado finito que operam no modo fundamental, portanto tem restrições temporais que os tornam menos robustos [15,36]. No exemplo mostrado em Fig. 13 temos os macromódulos *Decisão*, *Seqüência* e *Enquanto*.

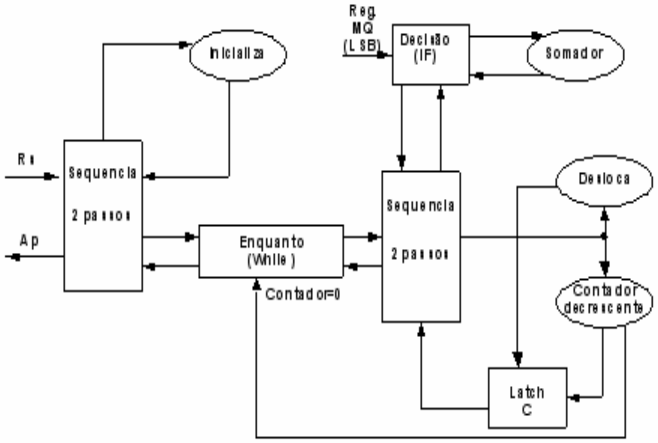

Fig. 13. Composição macromódulos: multiplicação [36].

III. ARQUITETURA MICROPIPELINE LINEAR PARA FPGA

As propostas para *micropipeline* na sua maioria são voltadas para registradores baseados em latches e os controladores são implementados com lógica *full-custom*. A nossa proposta voltada para FPGAs opera no protocolo *handshaking* de quatro fases. Ela procura aproveitar as características das macro-células das FPGAs que são ricas em flip-flops D (FFs) que possuem o sinal de *Reset* assíncrono. Figura 14 mostra a nossa arquitetura QDI e as Fig. 15,16,17 mostram respectivamente a especificação GMR a síntese e o circuito resultante do elemento de controle. Figura 18 mostra a nossa versão para FPGAs da arquitetura MOUSETRAP.

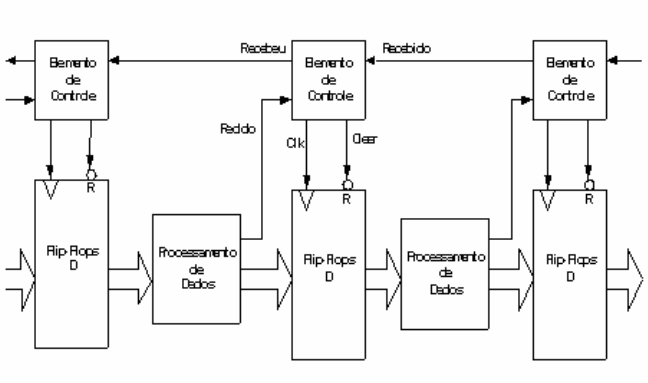

Fig. 14. Micropipeline linear voltada para FPGA.

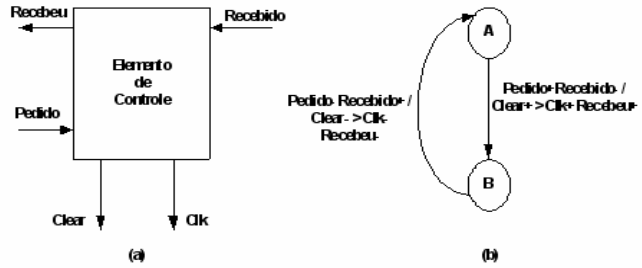

Fig. 15. Elemento de controle: a) estrutura lógica; b) grafo multi-rajada.

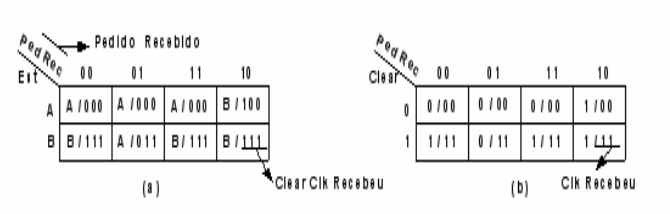

Fig. 16. Especificação: a) tabela de fluxo de estados; b) tabela de fluxo de estados codificada.

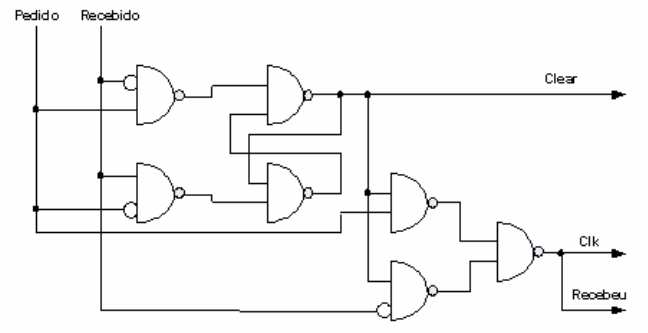

Fig. 17. Elemento de controle: circuito lógico.

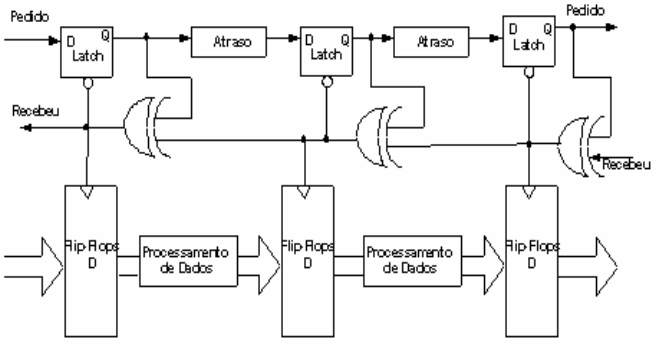

Fig. 18. Micropipeline MOUSETRAP: uma versão para FPGA.

IV. APLICAÇÃO

Para ilustrar a nossa arquitetura QDI usamos o projeto do filtro FIR de terceira ordem na versão assíncrona. A síntese comportamental do filtro FIR foi realizada usando cinco unidades funcionais dual-rail (três multiplicadores e dois somadores) que necessitou de três estágios. Figura 19 mostra a descrição RTL *pipeline* do nosso projeto.

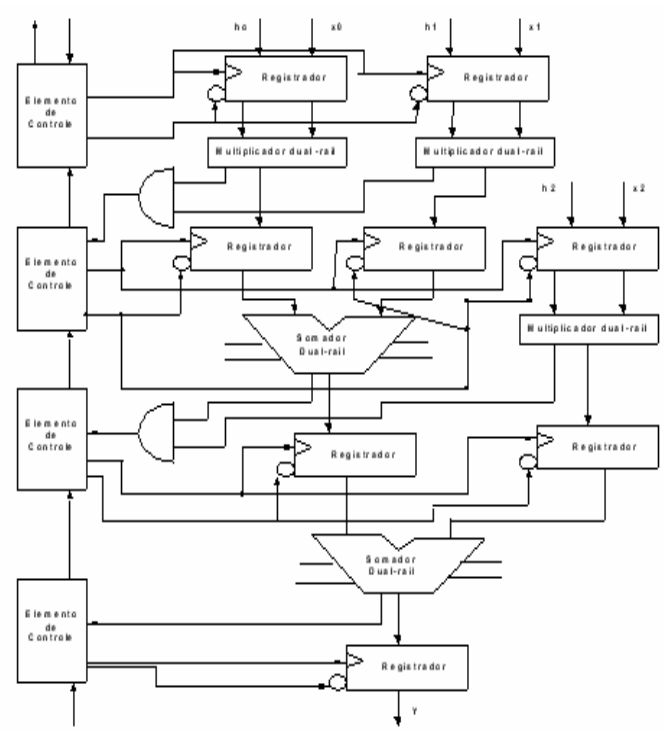

Fig. 19. Descrição RTL *micropipeline*: Filtro FIR de terceira ordem.

V. DISCUSSÃO

O projeto digital que estamos tratando têm duas condições iniciais, o ambiente embarcado e a implementação em FPGAs. Para muitas aplicações os sistemas embarcados podem ser descritos como um conjunto de controladores e *data-paths*, como mostrado em [37,38]. Tendo necessidade de micro-controladores, Furber et. al. [39] mostra a viabilidade do projeto assíncrono, portanto arquiteturas embarcadas podem ser implementadas no paradigma assíncrono.

Os três estilos apresentados para síntese de sistemas assíncronos podem usar a opção *single-rail*. Esta opção reduz área, potência dissipada e complexidade de projeto, mas carreta a introdução de elementos de atraso, porque o circuito de detecção de termino de operação para UF *single-rail* é muito complexo [15]. A introdução de elementos de atraso acarreta dois problemas: **a)** viola a classe QDI que possui propriedades interessantes já discutidas para a área embarcada; **b)** a inserção de elementos de atraso em FPGAs é difícil porque o compilador procura eliminar redundância. A solução é usar latches ou flip-flops para gerar o elemento de atraso.

A síntese *dual-rail* aumenta área e potência dissipada, mas preserva as propriedades da classe QDI. Os três estilos com a opção *dual-rail* têm as seguintes característica: o estilo decomposição (controlador+data-path) tem uma

tendência de aumentar as variáveis envolvidas (termino de operação) tornando o controlador mais complexo. Dos três estilos o de composição tem tendência de maior área e potência dissipada e menor desempenho [15,38]. Para implementar sistemas assíncronos em FPGA comerciais o estilo *micropipeline* é o mais adequado, porque reduz drasticamente a complexidade do controlador. Controladores assíncronos complexos ao serem mapeados em FPGAs podem acarretar problemas de risco. Stevens et. al. [40] mostra que as ferramentas CAD do mundo síncrono podem ser usadas em varias partes do projeto.

VI. CONCLUSÃO

Neste artigo mostramos a viabilidade do paradigma assíncrono quando usado no ambiente embarcado e ser implementado em dispositivos programáveis do tipo FPGAs baseadas em LUTs. Também apresentamos duas arquiteturas *micropipeline* voltadas para FPGAs. As duas arquiteturas aproveitam os recursos que as macro-células disponibilizam. Os nossos elementos de controle são de baixa complexidade, são facilmente mapeados, roteados e preservam as condições livres de risco. Trabalhos futuros aplicar e simular as duas novas arquiteturas como comportamento (corretude), desempenho e potência dissipada.

REFERÊNCIAS

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

[2] 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.

[3] P. P. Czapski and A. Sluzek, "A Survey on System-Level Techniques for Power Reduction in Field Programmable Gate Array (FPGA)-Based Devices", The Second Int. Conf. on Sensor Technologies and Applications, pp.319-327, 2008.

[4] Altera Corporation, 2009, www.altera.com.

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

[6] Li-Chuan Weng, X. J. Wang, and Bin Liu, "A Survey of Dynamic Power Optimization Techniques," Proc. Of the 3rd IEEE Int. Workshop on System-on-Chip for Real-Time Applications, pp. 48-52, 2003.

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

[8] S. M. Nowick, "*Automatic Synthesis of Burst-Mode Asynchronous Controlle*r", PhD thesis, Stanford University, 1993.

[9] K. Y. Yun, "*Synthesis of Asynchronous Controller for Heterogeneous*", PhD thesis, Stanford University, 1994.

[10] D. L. Oliveira, "*Miriã: Uma Ferramenta para a Síntese de Controladores Assíncronos Multi-Rajada*", Tese de doutorado, EPUSP, 2004.

[11] C. J. Myers and T. H.-Y. Meng, "Synthesis of Timed Asynchronous Circuits", *IEEE Trans. on VLSI Systems*, 1(2), pp.106-119, June 1993.

[12] J. Martin, "Compiling Communication to Delay-Insensitive VLSI Circuits", *Distributed Computing*, 1(4), pp.226-234, December 1986.

[13] J. Martin, "The Limitations to Delay Insensitive in Asynchronous Circuits," 6th MIT Conference on Advanced Research in VLSI Processes, pp.263-277, 1990.

[14] T. –A. Chu, "*Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications*", PhD thesis, Dept. of EECS, MIT, June 1987.

[15] J. Sparzo and S. Furber, "*Principles of Asynchronous Circuits Design*", Kluwer Academic Publishers, 2001.

[16] D. J. Barnhart, et. al. "Radiation Hardening by Design of Asynchronous Logic for Hostile Environments", *IEEE Journal of Solid-State Circuits*, vol. 44, No. 5, pp.1617-1628, May 2009.

[17] W. Hardt, et. al., "Architecture Level Optimization for Asynchronous IPs", Proc. 13th Annual IEEE Int. Conf. ASIC/SOC, pp.158-162, 2000.

[18] L. Zhenglin, et. al. "A High-security and Low-power AES S-Box Full-custom Design for Wireless Sensor Network", Int. Conf. Wireless Comu, Net. And Mobile Computing, pp.2499-2502, 2007.

[19] D. Shang, et. al., "High-security asynchronous circuit implementation of AES", *IEE Proc. Comput. Digit. Tech*. vol. 153, No. 2, March, 2006, pp.71-77.

[20] T.N. Prabakar, et. al., "FPGA Based Asynchronous Pipelined Multiplier with Intelligent Delay Controller", International SOC Design Conference, pp.304-309, 2008.

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

[22] 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.

[23] Payne, R., "Asynchronous FPGA architectures," *IEE Proc. Comp. Digit. Tech*., vol.143, no.5, pp. 282-286, September 1996.

[24] N. Huot, et. al., "FPGA architecture for multi-style asynchronous logic," Proc. Of the Design, Automation and Test in Europe Conference and Exhibition, pp. 32-33, 2005.

[25] P. Sigel, G. De Micheli and D. Dill, "Decomposition methods for library binding of speed-independent," Proc. Int. Conf. Computer-Aided Design, pp.558-565, 1994.

[26] 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, 2007.

[27] L. Plana and S. M. Nowick, "Architectural Optimization for Low-Power Nonpipelined Asynchronous systems", *IEEE Trans. on VLSI Systems*, vol.6 no.1, pp.56-65, March 1998.

[28] N. Hamada, et. al., "A Behavioral Synthesis Method for Asynchronous Circuits with Bundled-data Implementation (Tool Paper)", 8th Int. Conf. on Application of Concurrency to System Design, pp.50-55, 2008.

[29] 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.

[30] Kondratyev, "Design of Asynchronous Circuits Using Synchronous CAD Tools", IEEE Design & Test of Computers, 19,(4), pp.107-117, July-August, 2002.

[31] D. Sokolov, et. al. "STG optimization in the direct mapping of asynchronous circuits", Proc. Design Automation and Test in Europe, pp. 932-937, March 2003.

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

[33] 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, 2007.

[34] T. Chelcea, et al., "A Burst-Mode Oriented Back-end for the Balsa Synthesis System," Proc. Design, Automation and Test in Europe (DATE), pp.330-337, 2002.

[35] 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.

[36] S. H. Ungle, "A Building Block Approach to Unclocked Sytems", Proc. 26th Annu. Hawaii Int. Conf. Systems Sciences, vol.1, pp. 339-348, 1993.

[37] H. Hsieh, F. Balarin et. al. "Synchronous approach to the Functional Equivalence of Embedded System Implementations," *IEEE Trans. On CAD of Int. Circuits and Systems*, vol.20, no.8, pp.1016-1033, August 2001.

[38] L. Jozwiak, et al., "Multi-objective Optimal Controller Synthesis for Heterogeneous embedded Systems," Int. Conf. on Embedded Computer Systems: Architectures, Modeling and Simulation, pp. 177-184, 2006.

[39] S. B. Furber, et. al., "AMULET2e: An Asynchronous Embedded Controller", *Proc. Of the IEEE*, vol. 87, no. 2, pp. 243-256, February 1999.

[40] K. S. Stevens, Y. Xu, and V. Vij, "Characterization of Asynchronous Templates for Integration into Clocked CAD Flows", 15th IEEE Symposium on Asynchronous Circuits and Systems, pp.151-161, 2009.