Uma Abordagem Prática para Projeto Otimizado de Sistemas Digitais Assíncronos

Kledermon Garcia[1,2], Duarte L. Oliveira[1], 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* — Este trabalho apresenta um método para síntese otimizada de sistemas digitais assíncronos. A proposta usa o estilo de projeto conhecido como decomposição (*data-path* + controlador), que gera uma descrição intermediária nível RTL (*Register Transfer Level*). O método proposto emprega a especificação modo-*burst* estendido para descrever o controlador. Os sistemas assíncronos sintetizados pelo método proposto são capazes de operar no "protocolo *handshake* de duas fases", permitindo um melhor desempenho. Neste método a decomposição é implementada por *bundled-data*, usando, portanto, componentes do paradigma síncrono. Através de um caso de estudo, mostramos a simplicidade e a eficiência do método proposto.**

***Palavras-Chave* — Lógica assíncrona, síntese comportamental, programação linear inteira, controlador modo-*burst* estendido.**

I. INTRODUÇÃO

Sistemas digitais síncronos usam um sinal de relógio global para sincronizar as operações. Este sinal é uma das principais causas de ruído, emissões eletromagnéticas e consumo de potência. A definição da distribuição do sinal de relógio é uma tarefa complexa devido a sua defasagem ao longo do circuito (*clock skew*).

Sistemas digitais assíncronos não apresentam *clock skew* e consumo de energia é menor quando comparado com sistemas síncronos equivalentes. Circuitos assíncronos são mais robustos ao ruído e apresentam uma menor emissão eletromagnética [1,2]. A principal desvantagem dos circuitos assíncronos é a dificuldade de serem projetados livres de risco (*hazard*) e livres de corrida crítica [1,2]. Outro grande problema é a falta de ferramentas confiáveis para seu o projeto. Recentemente, tem havido várias tentativas para demonstrar as vantagens potenciais dos circuitos assíncronos sobre os circuitos síncronos [3-5] e alguns métodos para projeto de sistemas digitais assíncronos têm sido desenvolvidos com sucesso [6].

Kledermon Garcia, kedermonkg@iae.cta.br, Duarte L. Oliveira, duarte@ita.br, Tel +55-12-3947-6813, Fax +55-12-3947-6930, 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.

Neste artigo propomos um método para projeto de sistemas assíncronos otimizados no estilo decomposição *bunbled-data*, onde o controlador é descrito na especificação XBM (*extended burst-mode*). O método proposto tem as seguintes características: a) sintetiza o sistema na arquitetura alvo (ver Fig. 1); b) permite a interação com outros módulos no protocolo de 2-Fases (interação ponto a ponto), podendo ser uma estrutura em anel (ver Fig. 2); c) o projeto é todo baseado em portas básicas; d) na síntese comportamental o escalonamento das operações é realizado no paradigma assíncrono; e) elimina as transições de estado mortas que são intrínsecas ao protocolo de 4-Fases.

Com exemplo de aplicação, é apresentado o solucionador de uma equação diferencial de segunda ordem pelo método de Euler. Os resultados obtidos foram superiores a outras soluções assíncronas do mesmo problema. Em [3], a solução usa controle distribuído e portas complexas (circuitos transistorizados). A proposta de [7] emprega o escalonamento de operações do paradigma síncrono e, portanto, a solução obtida é sub-ótima. A proposta de [8] usa o protocolo de 4-Fases, que tem um *overhead* na comunicação, isto é, contém transições de estado mortas.

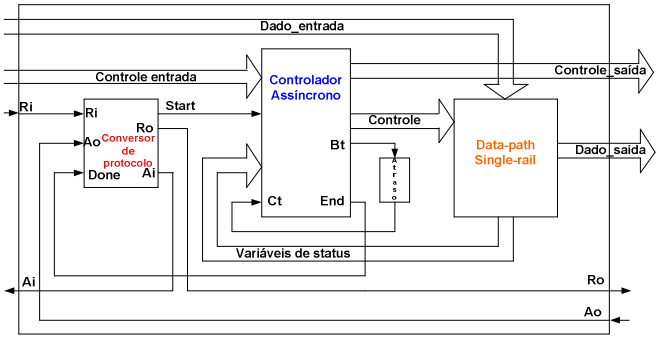

Fig. 1. Arquitetura alvo de [9].

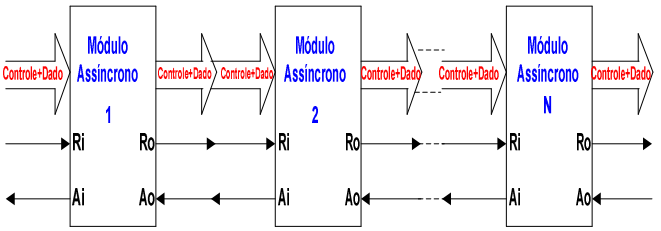

Fig. 2. Sistema assíncrono ponto-a-ponto.

II. SISTEMAS ASSÍNCRONOS

Um sistema digital assíncrono pode ser projetado em quatro diferentes estilos: decomposição; *micropipeline*; composição com macromódulos; e de-sincronização. No primeiro, a decomposição controlador + *data-path*, pode levar a uma otimização ótima [10-15]. O segundo estilo é interessante para aplicações com natureza pipeline, como microprocessadores, filtros digitais, etc. [16]. O terceiro estilo, também conhecido como método por tradução, é vantajoso em aplicações complexas, mas resulta em circuitos com baixa otimização [6,17-20]. No último estilo, de-sincronização, todo o projeto é realizado no paradigma síncrono, e a conversão gera um alto *overhead*, com consequente aumento de área e degradação do desempenho [21]. Entre os quatro estilos, o estilo decomposição é muito promissor devido à sua similaridade com o paradigma síncrono (ver Fig. 3).

Estilo decomposição para projeto assíncrono

Sistemas digitais assíncronos operam "por eventos", eles não têm um sinal global para sincronizar as operações. A sincronização dos sinais é realizada por um protocolo *handshake* (reconhecimento) e o sistema pode ser visto como um bloco funcional que interage com o ambiente através deste tipo de protocolo [1].

O estilo de projeto conhecido como "decomposição" segue o procedimento clássico dos métodos de projeto VLSI (ver Fig. 4). Os projetos podem ser sintetizados em diferentes classes de circuitos assíncronos, onde a classe define em qual modelo de atraso o circuito irá operar corretamente e o modo de comunicação com o ambiente [1,2]. Este estilo de projeto tem duas variantes. Na primeira, o projeto envolve *data-path* assíncrono e por isso não apresenta qualquer elemento de atraso. O *data-path* assíncrono, entretanto, envolve componentes *dual-rail*, que apresentam um custo elevado de implementação [22]. Na segunda variante, "por decomposição", o *data-path* é implementado no protocolo "*bundled-data*", que envolve componentes *single-rail* (componentes do paradigma síncrono), e um ou mais elementos de atraso são inseridos no projeto assíncrono para que haja uma comunicação correta entre o controle e o data-path (requisito do modo fundamental) [7,8,10-14]. Esta variante leva a circuitos com baixo consumo de energia, menor área e projeto mais simples, quando comparado com a primeira variante.

Duas especificações são comumente usadas para descrever controladores assíncronos, a especificação modo *burst* estendido (*extended burst-mode* – XBM) proposta por Yun e Dill [23,24], e a especificação grafo de transição de sinais (*signal transition graph* – STG) proposta por Chu [25]. A primeira é a mais indicada para controladores que interagem com ambientes do tipo *bundled-data*, isto é que satisfazem o modo fundamental. A segunda é mais adequada para controladores que interagem com ambientes concorrentes. Para síntese automática, há a ferramenta 3D que sintetiza os controladores XBM [24], e a ferramenta Petrify sintetiza os controladores STG [26].

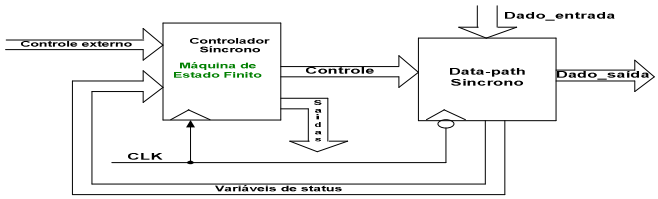

Fig. 3. Arquitetura sincrona RTL.

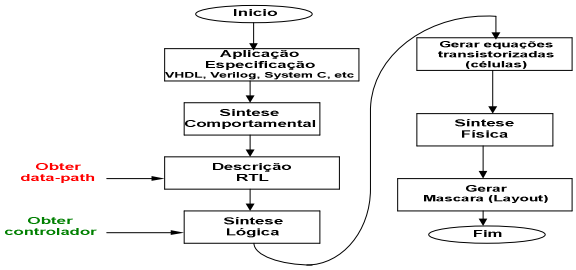

Fig. 4. Método para projeto VLSI.

Um sistema digital assíncrono é visto como um módulo funcional que se comunica com o ambiente (um outro módulo) no protocolo *handshake* (ver Fig. 5). Este protocolo usa os sinais *request* (*Ri*, *Ro*) e *acknowledge* (*Ai*, *Ao*) para realizar a comunicação. Esta comunicação pode ser no protocolo *handshake* de quatro fases, onde os dois sinais usam apenas uma borda (ver Fig. 5a), ou no protocolo de duas fases, onde os dois sinais usam as duas bordas na comunicação (ver Fig. 5b).

Existem diferentes abordagens propostas para o estilo decomposição na síntese de sistemas assíncronos. Estas abordagens ou levam a uma otimização sub-ótima do sistema, porque usam algoritmos do paradigma síncrono ou usam procedimentos mais refinados, que dificultam o projeto.

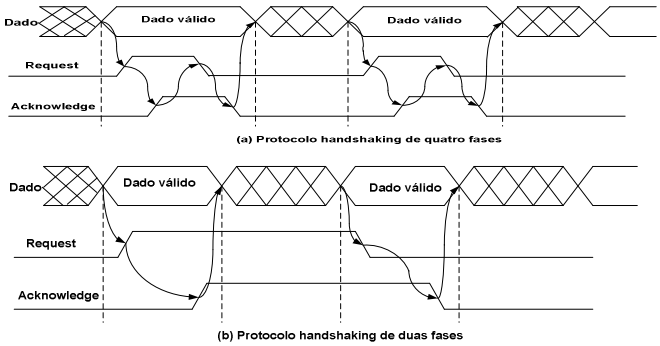

Fig. 5. Protocolo *handshake*.

III. SÍNTESE COMPORTAMENTAL ASSÍNCRONA

A síntese comportamental ou síntese de alto nível parte de uma especificação do tipo VHDL ou similar e gera uma descrição otimizada RTL. A Fig. 6 mostra os passos da síntese comportamental do método proposto. O método extrai da especificação o grafo de fluxo de dados e de controle [27] e aplica os algoritmos de escalonamento assíncrono, assinalamento de registradores e alocação de unidades funcionais.

Escalonamento assíncrono

A tarefa de escalonamento é um problema de natureza combinatória, portanto é um problema NP-completo. Ela define o tempo inicial das operações sobre as restrições de recursos ou tempo. Através do escalonamento obtemos o número de operações que serão processadas por passo de controle, a sequência de operações e o número de passos de controle que a aplicação irá usar. Os algoritmos de escalonamento usam uma estrutura intermediária denominada grafo de fluxo de dados e de controle (GFDC) para descrever a sequência das operações e gerar um GFDC escalonado, isto é, definindo o número de passos de controle e as operações a cada passo. Muitos métodos propostos para síntese de sistemas assíncronos usam algoritmos de escalonamento do paradigma síncrono [7,10,11,14], com soluções sub-ótimas, porque o passo de controle representa um ciclo do *clock*, não existente paradigma assíncrono.

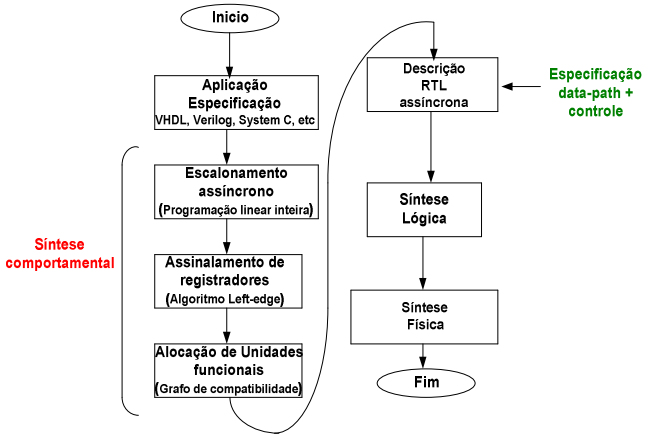

Fig. 6. Método proposto: síntese comportamental.

O nosso método usa o algoritmo de escalonamento assíncrono exato proposto em [8] que é baseado em programação linear inteira (PLI). O algoritmo de [8] parte das restrições do número de unidades funcionais (UFs) analisa os atrasos das UFs e distribui as operações no menor número de passos. O escalonamento é definido pelas equações (1), (2), (3) e (4), onde a resolução deste sistema pode ser feita através da ferramenta Matlab ou pela ferramenta apresentada em [28].

A função objetivo para escalonamento com restrição de recursos é:

$$Minimize: \sum_{r \in R} Custo_r * Num_r \quad (1)$$

Onde, r representa um recurso de uma dada biblioteca R, $Custo_r$ representa o custo do recurso r, e Num_r representa o número de recursos do tipo r. A função objetivo minimiza o custo total dos recursos do *data-path* sobre restrição do tempo. Há três tipos de restrição:

a) Restrição de assinalamento do nó

Cada nó i deve ser escalonado exatamente em um passo $l \in Passo_i$. Esta restrição é representada como:

$$\sum_{l \in Si} x_{i,l} = 1, \ \forall i \in N \quad (2)$$

b) Restrição de dependência

Para cada arco direcionado (i,j) do GFD, devem ser satisfeitas as restrições de dependência:

$$\sum_{l \in Passo_j} t_l \times x_{j,1} - \sum_{l \in Passo_i} t_1 \times x_{i,1} - d_i \geq 0, \ \forall (i,j) \in E \quad (3)$$

c) Restrição de assinalamento de recurso

O número de operações executadas pelo recurso r no passo l deve ser igual a *Num* ou menor. A restrição de assinalamento de recurso é representada como segue:

$$\sum_{i \in N_r} x_{i,j} - Num_r \leq 0, \ \forall 1 \in Passo_r, \ \forall r \in R \quad (4)$$

Onde: N_r é o conjunto de nós que está relacionado com o recurso r e Num_r é um valor constante.

Assinalamento de registradores e alocação de Ufs

A tarefa de assinalamento de registradores procura compartilhar as variáveis usando o conceito de "tempo de vida" da variável. Esta tarefa também é um problema NP-completo. O nosso método parte do grafo de compatibilidade e usa um algoritmo heurístico para encontrar sub-grafos que é muito simples e bastante eficiente [29]. O assinalamento das variáveis em um conjunto mínimo de registradores influencia diretamente as conexões, isto é no número de multiplexadores. A tarefa de alocação de unidades funcionais está relacionada com a escolha das operações com o operador (unidade funcional). O nosso método usa o grafo de compatibilidade para realizar esta tarefa. O algoritmo procura particionar o grafo em sub-grafos que tem o menor custo. Diferentes algoritmos existem para encontrar estes sub-grafos, por exemplo, algoritmos de coloração de grafo [29]. O nosso método usa o algoritmo de clique [29].

IV. SÍNTESE LÓGICA ASSÍNCRONA

O controlador assíncrono é sintetizado no estilo de síntese lógica, que permite obter um controlador assíncrono otimizado. Fig. 7 mostra os três passos para a síntese lógica assíncrona. O nosso controlador assíncrono é descrito na especificação XBM, e obedece ao modelo de atraso *bounded gate and wire delay*, onde os atrasos são delimitados em mínimo e máximo. A interação com o ambiente ocorre no modo fundamental generalizado (MFG). No MFG, a cada ativação de uma nova entrada *burst* o controlador deve estar estabilizado, isto é sem atividade lógica. No nosso sistema assíncrono, tanto o *data-path* como o controlador opera no MFG; assim, na interação entre os dois blocos, eles devem estar estabilizados. O controlador XBM extraído pelo nosso método é sintetizado pela ferramenta 3D [24]. Esta ferramenta realiza dois passos, que são assinalamento de estados e minimização lógica. Ela implementa seus circuitos na arquitetura de Huffman com saída realimentada.

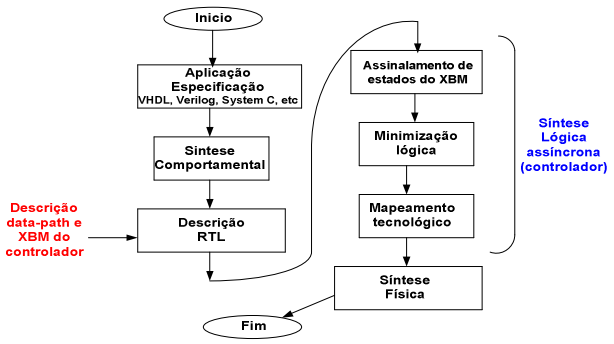

Fig. 7. Síntese lógica: controlador assíncrono.

Modo-burst estendido – XBM

A especificação *data-drive*, proposta por Al Davis et al. [30] e formalizada por Nowick [31] como *burst-mode* (BM), é representada por um grafo na qual os vértices representam estados estáveis, enquanto os arcos representam as transições de estado. Um estado inicial deve existir. Yun e Dill [23,24] propuseram uma extensão no BM, denominada especificação modo-burst estendido (*extended burst-mode* – XBM) adicionando duas características: sinais irrelevantes (*direct don't-care*) que permite um sinal de entrada seja ativado concorrentemente com os sinais de saída; e sinais condicionais que são sinais sensíveis ao nível com comportamento não monotônico) [23]. As restrições do BM foram generalizadas para permitir a extensão proposta em [23,24].

Nós ilustramos esta especificação com o benchmark Biu-dma2fifo da HP. Fig. 8 mostra uma especificação XBM com 4 entradas (*cntgt1,dackn, fain,ok*) , 2 saídas (*dreq,frout*) e o estado inicial 0. A descrição *fain+ dackn+/ frout+ na* transição 2→3 significa que a saída (frout: 0→1) vai seguir a entrada *burst* (*fain:* 0→1 *AND dackn:* 0→1). O sinal sensível ao nível *cntgt1* é usado para descrever exclusão mútua entre as transições 1→5 e 1→2. O sinal irrelevante *fain** na transição 0→1 significa que o sinal *fain* tanto pode alterar seu valor ou permanecer em seu valor antigo.

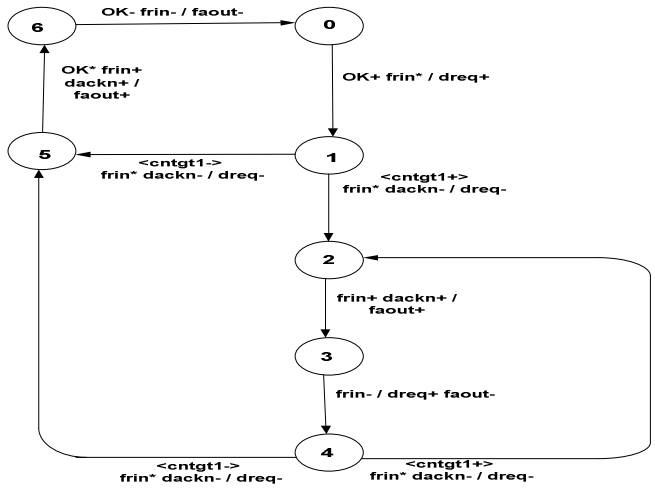

Fig. 8. Síntese lógica: controlador assíncrono Biu-dma2fifo.

V. MÉTODO: SISTEMAS ASSÍNCRONOS

O método proposto segue os passos tradicionais da síntese comportamental e lógica, e implementa na arquitetura alvo da Fig. 1. A arquitetura usa o conversor de protocolo de [9], que permite com que a comunicação com o ambiente externo seja no protocolo de 2-Fases (ver Fig. 5). Fig. 9 mostra o fluxograma do método, que pode ser dividida em cinco passos:

1. Realiza a síntese comportamental assíncrona (ver seção II); gera o GFDC escalonado e o assinalamento de registradores e operadores (UFs).
2. A partir do passo (1) gerar o data-path inicial e a descrição do controlador XBM voltado para o protocolo de 4-Fases.
3. Transformar a especificação XBM de 4-Fases do controlador em 2-Fases; identificar e eliminar as transições de estado mortas. Uma transição de estado no XBM é definido como morta, quando a saida *burst* rotulada na transição não executa nenhuma operação, apenas muda de fase (chamada fase zero).
4. A partir do XBM de 2-Fases restruturar o data-path, substituindo componentes SET por componentes DET [9].
5. Sintetizar a especificação XBM de 2-Fases do controlador utilizando a ferramenta 3D.

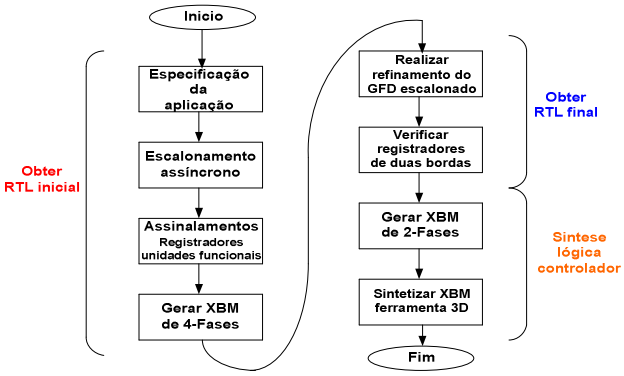

Fig. 9. Método por decomposição proposto.

VI. CASO DE ESTUDO

O método proposto será ilustrado para o solucionador da equação diferencial de 2ª ordem: $y''+3xy'+3y=0$ segundo o método de Euler. Fig. 10 mostra o pseudocódigo do método para solução [27].

O primeiro passo realiza a síntese comportamental, onde Fig. 11 mostra o sistema de equações PLI e a sua solução, que emprega dois multiplicadores e uma ULA. Fig. 12 mostra o GFDC escalonado final, com sete passos de controle. Fig. 12 também mostra dois diferentes tempos nos passos de controle, que são 2,8ns e 4,7ns. Fig. 13 mostra o assinalamento de registradores usando grafo de compatibilidade, no caso são oito SET-registradores. Fig. 14 mostra o assinalamento das Ufs, no caso dois multiplicadores e uma ULA, que realiza as

operações de adição, subtração e comparação. O *data-path* inicial também contém oito bancos de multiplexadores.

O segundo obtém a descrição do XBM de 4-Fases, que contém 22 estados, 23 transições de estado, 5 sinais de entrada e 24 sinais de saida.

O terceiro passo identifica as transições mortas gerando a especificaçao XBM de 2-Fases; o novo XBM contém 11 estados, 12 transições de estado, 4 sinais de entrada e 21 sinais de saida (ver Fig. 15).

O quarto passo reestrutura o *data-path*, que passa a ter seis registradores, sendo que os registradores R2 e R5 são DET (ver Fig. 16). O esquema geral do S_Euler_EDA está mostrado em Fig. 17, que resulta na inserção de dois elementos de atraso, relacionados com os dois tempos dos passos de controle.

O passo final sintetiza o controlador na ferramenta 3D, na arquitetura de Huffman com saida realimentada. O circuito lógico é formado por 23 funções Booleanas soma de produto, onde duas variáveis de estado foram introduzidas. As 23 funções contêm 68 produtos e 179 literais.

```
Procedure S_Euler_ED
{
  read( a, dx, x, y, u);
  repeat {
          x1 = x+dx;
          u1 = u - (3* x* u* dx) - (3* y* dx);
          y1 = y + u* dx;
           c = x1 < a;
            x = x1; u = u1; y = y1;
         }
  until (c );
  write (y);
}
```

Fig. 10. Pseudocódigo do solucionador da equação diferencial.

a) Função Objetivo:
Minimizar: $5num_{mult} + num_{alu}$

b) Restrições de Recurso

$X_{1,1} + X_{2,1} + X_{3,1} - num_{mult} \le 0$
$X_{1,3} + X_{2,3} + X_{3,3} + X_{6,3} + X_{7,3} - num_{mult} \le 0$
$X_{6,5} + X_{7,5} - num_{mult} \le 0$
$X_{4,4} + X_{10,4} - num_{alu} \le 0$
$X_{4,6} + X_{5,6} + X_{9,6} + X_{10,6} - num_{alu} \le 0$
$X_{9,7} + X_{10,7} - num_{alu} \le 0$
$X_{5,5} + X_{8,5} - num_{alu} \le 0$
$X_{4,1} - num_{alu} \le 0$
$X_{5,3} - num_{alu} \le 0$
$X_{5,5} + X_{8,5} - num_{alu} \le 0$
$X_{10,2} - num_{alu} \le 0$

c) Restrições de Dependência:

$75X_{5,3} + 150X_{5,5} + 178X_{5,6} - 0X_{1,1} - 75X_{1,3} \ge 75$
$75X_{6,3} + 150X_{6,5} - 0X_{1,1} - 75X_{1,3} \ge 75$
$75X_{6,3} + 150X_{6,5} - 0X_{2,1} - 75X_{2,3} \ge 75$
$75X_{7,3} + 150X_{7,5} - 0X_{3,1} - 75X_{3,3} \ge 75$
$28X_{10,2} + 103X_{10,4} + 178X_{10,6} + 225X_{10,7} - 0X_{4,1} - 103X_{4,4} - 178X_{4,6} \ge 28$
$150X_{8,5} + 178X_{8,8} - 75X_{6,3} - 150X_{6,5} \ge 75$
$178X_{9,6} + 225X_{9,7} - 150X_{8,5} - 178X_{8,6} \ge 28$
$178X_{9,6} + 225X_{9,7} - 75X_{7,3} - 150X_{7,5} \ge 75$

d) Restrições de atribuição do nó:

$X_{1,1} + X_{1,3} = 1$
$X_{2,1} + X_{2,3} = 1$
$X_{3,1} + X_{3,3} = 1$
$X_{4,1} + X_{4,4} + X_{4,6} = 1$
$X_{5,3} + X_{5,5} + X_{5,6} = 1$
$X_{6,3} + X_{6,5} = 1$
$X_{7,3} + X_{7,5} = 1$
$X_{8,5} + X_{8,6} = 1$
$X_{9,6} + X_{9,7} = 1$
$X_{10,2} + X_{10,4} + X_{10,6} + X_{10,7} = 1$

e) Solução:

$num_{alu} = 1$
$num_{mult} = 2$
$X_{1,1}, X_{2,1}, X_{3,3}, X_{4,1}, X_{5,3}, X_{6,3}, X_{7,5}, X_{8,5}, X_{9,7}, X_{10,4}$

Fig. 11. Sistema de equações PLI e solução para S_Euler_EDA.

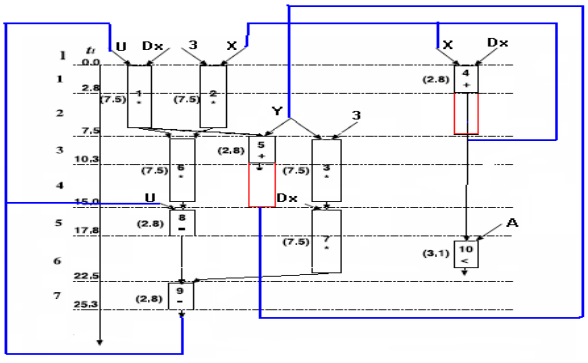

Fig. 12. Grafo de fluxo de dados e controle do S_Euler_EDA.

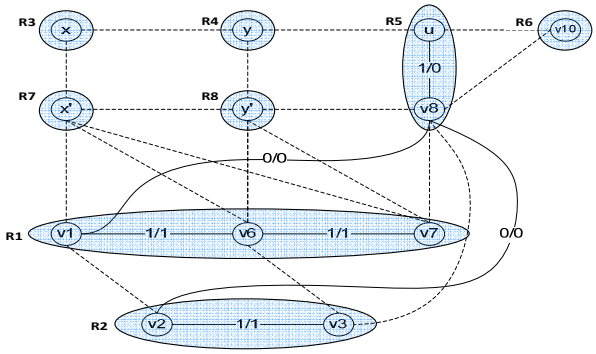

Fig. 13. Grafo de compatibilidade: assinalamento de registradores do S_Euler_EDA.

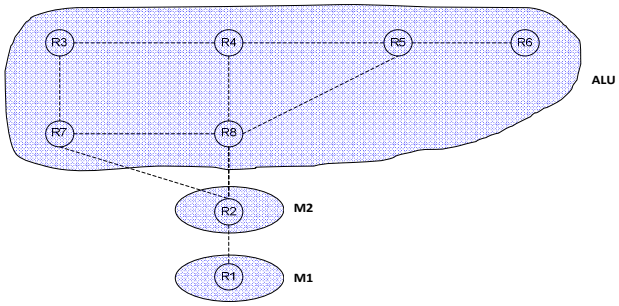

Fig. 14. Grafo de compatilidade: assinalamento UFs para S_Euler_EDA.

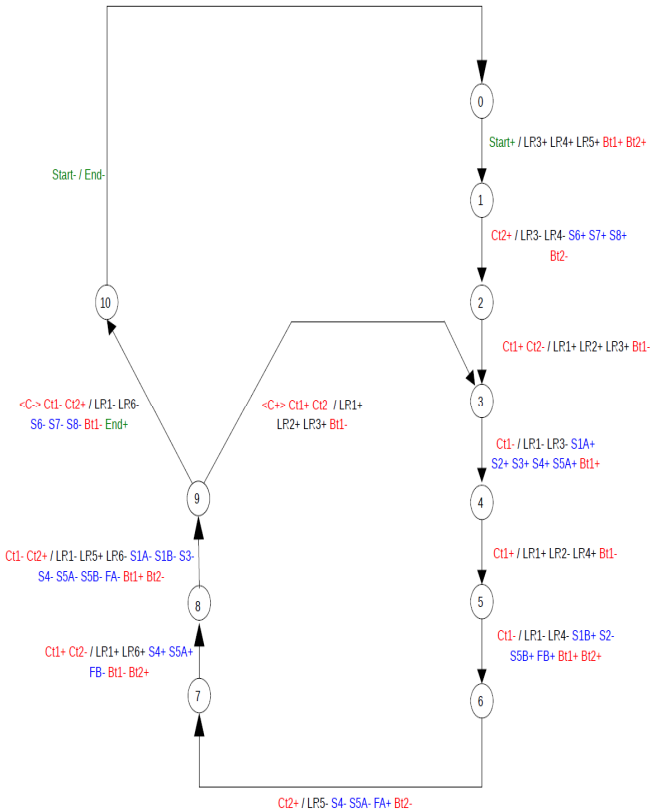

Fig. 15. Decrição do controlador XBM do S_Euler_EDA.

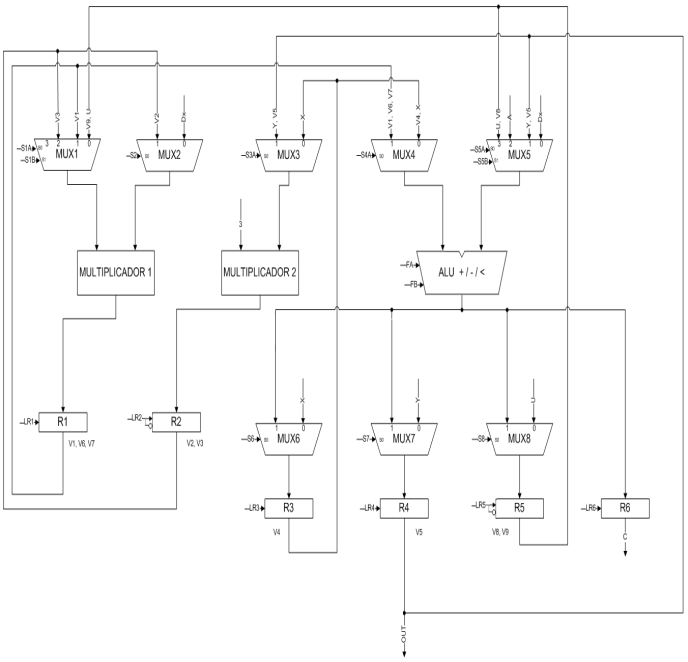

Fig. 16. Decrição data-path final do S_Euler_EDA.

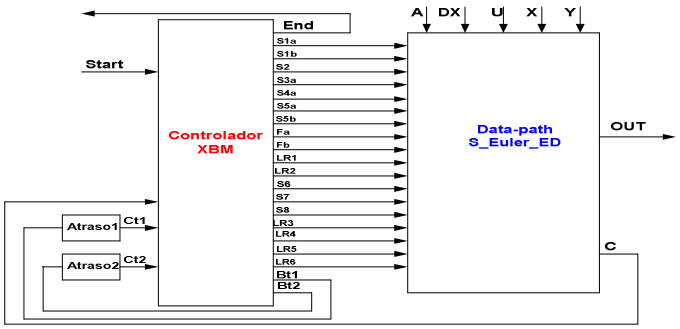

Fig. 17. Esquema geral: S_Euler_EDA.

VII. Resultados & Simulação

A solução encontrada na seção VI foi sintetizada e simulada para uma FPGA STRATIX II da Altera (EP2S15F484C3) [32]. Fig. 18 mostra as formas de onda livre de *hazard* e os resultados obtidos de uma simulação. Neste caso, as entradas são: *A*=6; *Dx*=2; *U*=7; *X*=3; *Y*=10, obtendo-se a solução final *Out*=142. Fig. 19 mostra a simulaçao da versão síncrona do projeto S_Euler_ED. A Tabela 1 mostra os resultados de área da três versões do S_Euler_ED, incluindo a versão assíncrona obtida pelo método do Saito et al. [8] e, o tempo de processamento, que é obtido de *Start* + → *End*+ para os dados de entrada citados. A solução proposta para S_Euler_EDA obteve uma redução média no tempo de processamento de 30%. No requisito area, entretanto, teve uma penalidade de 40% no número de LUTS e uma penalidade de 31% no número de *flip-flops*, quando comparado com a versão síncrona. Comparando com a versão assíncrona do Saito, a nossa proposta teve uma penalidade de 18% no número de LUTS e uma redução de 2% no número de *flip-flops*.

TABELA 1 RESULTADOS: PROJETOS

S_Euler_ED	tempo de processamento	Macrocélulas	
		Nro. LUTS	Nro. Flip-Flops
Síncrona	227,55ns	**241**	**164**
Saito [8]	219,74ns	328	247
Proposta	**155,22ns**	402	240

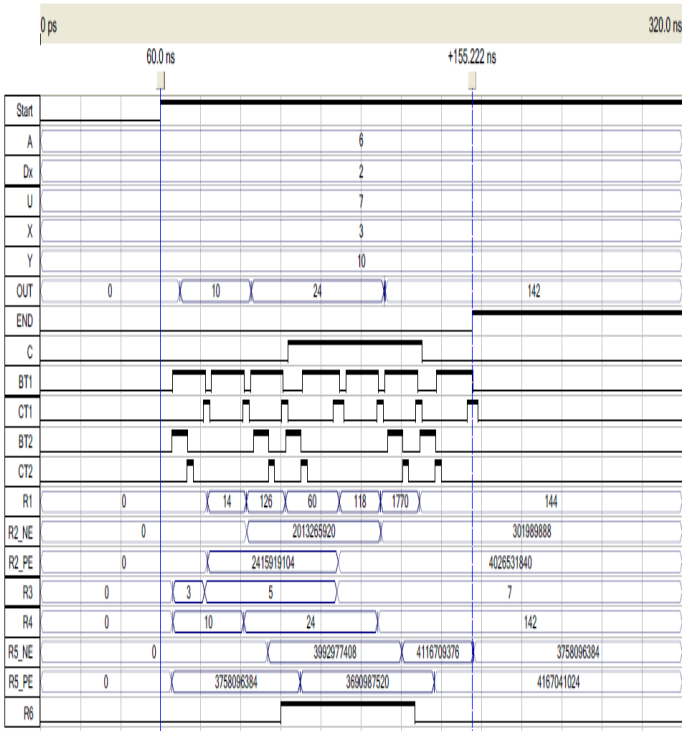

Fig. 18. Simulação: solucionador assíncrono da equação diferencial.

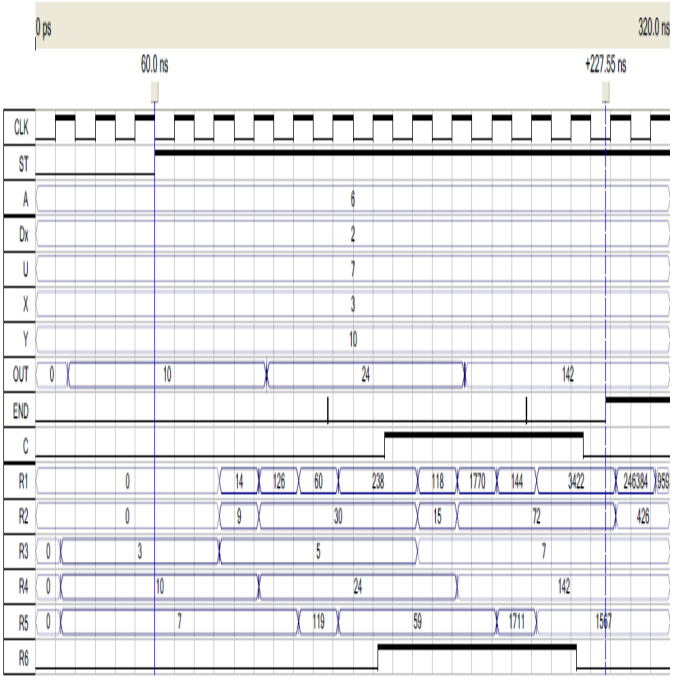

Fig. 19. Simulação: solucionador síncrono da equação diferencial.

VIII. CONCLUSÃO

Neste artigo apresentamos um método para projetar sistemas digitais assíncronos no estilo decomposição. Este estilo é familiar aos projetistas do paradigma síncrono. O método apresentado parte de um escalonamento assíncrono e usa componentes que operam nas duas bordas do sinal, permitindo uma redução significativa do número de transições de estado do XBM, o que acarreta um aumento de desempenho. Através de um caso bem conhecido que é S_Euler_SD, mostramos que o nosso método obtém resultado expressivo no tempo de processamento. Para trabalho futuro, é o desenvolvimento de uma ferramenta para o método proposto (síntese automática), permitindo aplicá-lo nos mais diferentes projetos.

REFERÊNCIAS

[1] J. Sparsø, "Current Trend in High-Level Synthesis of Asynchronous Circuits," Proc.16th IEEE Int. Conf. on Electronics, Circuits and Systems, pp.347-350, 2009.

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

[3] K. Y. Yun, et al., "The design and verification of high-performance low-control-overhead asynchronous differential equation solver," in *IEEE Trans. VLSI Syst.*, vol. 6, pp. 1-14, Dec. 1998.

[4] H. Van Gageldonk, et. al., "An asynchronous low-power 80c51 microcontroller," Proc. IEEE International Symposium on Advanced Research in Asynchronous Circuits and Systems, pp. 96-107, 1998.

[5] A. Bink and R. York, "ARM996HS: The First Licensable, Clockless 32-Bit Processor Core," *IEEE Micro*, vol 27, Issue 2, pp. 58-68, March-April 2007.

[6] D. Edwards and A. Bardsley, "Balsa: An asynchronous hardware synthesis language," *The Computer Journal*, vol. 45, no. 1, pp. 12 –18, Jan. 2002.

[7] S. Hiroshi, et al., "A Floorplan Method for Asynchronous Circuits with Bundled-data Implementation on FPGAs", Proc. IEEE Int. Symposium on Circuits and Systems (ISCAS), pp.925-928, 2010

[8] H. Saito, et al., "Scheduling Methods for Asynchronous Circuits with Bundled-Data Implementations Based on the Approximation of Start Times," *IEICE Trans. Fundamentals*, vol. E90-A, Nro. 12, pp. 27902799, December 2007.

[9] D. L. Oliveira, et al., "Synthesis of Asynchronous Digital Systems of High Performance using Simpler Approach," Proc. XVIII Iberchip Workshop, pp.74-79, 2012.

[10] B. M. Bachman, "*Architectural-Level Synthesis of Asynchronous Systems*," Master of Science, University of Utah, pp. 101, 1998.

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

[12] M. Lizuka, et al., "A Tool for the Design of Asynchronous Circuits with Bundled-data Implementation", Proc. IEEE 29th Int. Conf. on Computer Design (ICCD), pp.78-83, 2011.

[13] J. Hansen and M. Singh, "A Fast Branch-and-Bounded Approach to High-Level Synthesis of Asynchronous Systems", Proc. IEEE 16th Symposium on Asynchronous Circuits and Systems, pp.107-116, 2010.

[14] T. Konishi, et al., "A Control Circuit Synthesis Method for Asynchronous Circuits in Bundled-Data Implementation", Proc. 7th Int. Conf. on Computer and Information Technology, pp.847-852, 2007.

[15] N. Andrikos and L. Lavagno, "Optimal and Heuristic Scheduling Algorithms for Asynchronous High-Level Synthesis", Proc. 17th Symposium on Asynchronous Circuits and Systems, pp.13-21, 2011.

[16] Sutherland, I. E., "Micropipelines," *Commun. ACM*, vol. 32, no. 6, pp. 720–738, June 1989.

[17] S. F. Nielsen, J. Sparsø e J. Madsen, "Behavioral Synthesis of Asynchronous Circuits using Syntax Directed Translation as Backend," *IEEE Trans. on VLSI Systems*, vol. 17, Nro. 2, pp.248-261, February 2009.

[18] 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, March 2002.

[19] S. F. Nielsen, J. Sparsø, and J. Madsen, "Behavioral Synthesis of Asynchronous Circuits Using Syntax Directed Translation as Backend," *IEEE Trans. on Very Large Scale Integration (VLSI)*, vol. 17, nro. 2, pp. 248-261, February 2009.

[20] M. Sacker, et al., "Behavioral Synthesis System for Asynchronous Circuits," *IEEE Trans. on VLSI Systems*, vol. 12, Nro. 9, pp.978-994, September, 2004.

[21] 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.

[22] D. Sokolov, et al., "Design and Analysis of Dual-Rail Circuits for Security Applications," *IEEE Transactions on Computers*, vol. 54, no. 4, pp. 449-460, April 2005. .

[23] K. Y. Yun and D. L. Dill, "Automatic synthesis of extended burst-mode circuits: Part I (specification and hazard-free implementations)," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 101–117, Jan., 1999.

[24] _____,"Automatic synthesis of extended burst-mode circuits: Part II (automatic synthesis)," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 118–132, Jan., 1999.

[25] T. -A. Chu, "*Synthesis of Self-Timed VLSI Circuits from Graph-Theory Specifications,*" Ph.D. thesis, June, 1987, Dept. of EECS, MIT.

[26] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers," *IEICE Trans. Inf. Syst*., vol. E80-D, no. 3, pp. 315–325, Mar. 1997.

[27] G. De Micheli, "*Synthesis and Optimization of Digital Circuits*" McGraw Hill International Editions, 1994.

[28] "lp solve: A Mixed Integer Linear Programming (MILP) solver," available online at http://lpsolve.sourceforge.net/, 2013.

[29] D. D. Gajski, "*Principles of Digital Design*," Prentice Hall, 1997.

[30] A. L. Davis, et al., "A data-driven machine architecture suitable for VLSI implentation," In C.L. Seitz, editor, Proc. of the Caltech Conf. on Very Large Scale Integration, pp.179-194, 1979.

[31] S. M. Nowick, "*Automatic Synthesis of Burst-Mode Asynchronous Controllers,*" Ph.D. thesis, Stanford University, 1993.

[32] ALTERA Corporation-www.altera.com, 2013.