Projeto Prático de Controladores Assíncronos Voltados para Comunicação de Sistemas GALS

Duarte L. Oliveira[1], Eduardo Lussari[1], Lester A. Faria[1] e João Luis V. Oliveira[2]
[1]Divisão de Engenharia Eletrônica do Instituto Tecnológico de Aeronáutica – ITA – IEEA
Praça Marechal Eduardo Gomes, 50, CEP 12228-900, SJC, SP, Brasil
[2]Instituto Pesquisas Eldorado – Campinas, SP, Brasil

***Resumo* — Avanço da microeletrônica (tecnologia 35nm, 10^9 transistores – *Deep-sub-micron* – DSM) permite que sistemas digitais síncronos VLSI (*Very Large Scale Integrated*) sejam cada vez mais complexos, mas ao mesmo tempo enormes difuculdades de projeto aparecem relacionados com o sinal de *clock* global. A tecnologia DSM é sensível á interferência eletromagnética, alto consumo de potência, e o atraso nas linhas podem ser maiores que o atraso nas portas. O sinal de *clock* acarreta aumento da potência, alta interferência eletromagnética, e problemas de *clock* skew e distribuição do *clock*. Um estilo interessante para projetos nas plataformas VLSI-DSM e FPGA-DSM (*Field Programmable Gated Array*) devido à ausência do sinal de clock global é o paradigma GALS (*Globally Asynchronous, Locally Synchronous*). Atualmente, o grande problema na concepção de um sistema GALS mostra ser a interface assíncrona (IA). Existe um estilo típico de IA que é baseado em controladores assíncronos (chamados de *ports*), que permite uma melhor comunicação entre os módulos síncronos, mas os controladores *ports* estão sujeitos as dificuldades inerentes do paradigma assíncrono. Este artigo apresenta um método para síntese de controladores *ports* que visa reduzir as dificuldades de projeto, quando comparado com os outros métodos.**

***Palavras-Chave* — Grafo *Multi-Burst*, controladores *ports*, síntese lógica, interface assíncrona.**

I. INTRODUÇÃO

O avanço da microeletrônica segue a lei de Moore, que oferece aos projetistas de sistemas digitais VLSI (*Very Large Scale Integrated*) a possibilidade de construir cada vez mais, circuitos digitais mais complexos (tecnologia 35nm, 10^9 transistores – *Deep-sub-micron* – *DSM)*, mas ao mesmo tempo cria vários desafios, quando se constroem em tais estruturas [1,2]. A tecnologia DSM é sensível á interferência eletromagnética, alto consumo de potência, e o atraso nas linhas pode ser maior que o atraso nas portas [2]. Devido a estes problemas, o grande desafio é a distribuição do *clock* global através de todo o *chip* com defasagem mínima (*clock skew*) sem implicar em um grande esforço de projeto, área ocupada e aumento de potência [3].

Duarte L. Oliveira, duarte@ita.br, Tel +55-12-3947-6813, Fax +55-12-3947-6930, Eduardo Lussari, lussari@gmail.com.br, Lester A. Faria, lester@ita.br, João Luiz V. Oliveira, jaovillar@gmail.com - ITA - IEEA - Praça Marechal Eduardo Gomes, 50, Vila das Acácias - São José dos Campos - SP - Brasil.

Ao aumentar a complexidade dos sistemas digitais e da frequência do relógio, a metodologia totalmente sincronizada procura solucionar este grande desafio, que é o problema da fase defasada do *clock* e a distribuição do *clock* global, com um alto número de *buffers* e um cuidadoso projeto da arvore do *clock*, que introduz uma considerável área e um aumento de potência [4]. Em uma CPU de alto desempenho quase 40% do consumo da potência total do circuito é consumida pelo *clock* [5,6].

Metodologias de projeto assíncrono [7,8] podem naturalmente eliminar estes desafios ao remover o sinal do *clock* do projeto. No entanto, estes circuitos não são uma solução amplamente aceita, devido à falta de ferramentas CAD (*computer aided design*) para o projeto de circuitos assíncronos, dificuldades do projeto livre de risco (*hazard*) e pouca cultura no projeto assíncrono. Soluções intermediárias podem ser encontradas entre totalmente síncrono e totalmente assíncrono, a saber, a metodologia síncrona localmente e assíncrona globalmente (*Globally Asynchronous Locally Synchronous – GALS*). O termo GALS foi primeiramente usado por Chapiro em seu PhD [9]. Um sistema GALS consiste de módulos funcionais síncronos que se comunicam uns com os outros na forma assíncrona. Este paradigma é uma combinação dos paradigmas síncronos e assíncronos. Neste artigo referenciamos um sistema GALS como um sistema digital particionado em módulos funcionais síncronos que podem ser IPs (*Intellectual Property*) das mais diferentes empresas [10]. Eles possuem *clocks* próprios com frequências não relacionadas. Os módulos IP são pré-projetados, verificados, testados e possuem alto desempenho. Eles permitem reduzir custo e principalmente o tempo do projeto. Um esquema de comunicação assíncrona é usado para a comunicação entre os módulos com diferentes domínios de *clock*. Para manusear a comunicação assíncrona entre os módulos, um circuito de interface é adicionado em torno de cada módulo síncrono, gerando um invólucro assíncrono (*asynchronous wrapper*) [11]. Esta interface pode ser composta, por exemplo, de *clock* local, FIFO, controlador assíncrono de comunicação (*Inputs Ports, Output Ports*). Techan et al. [12] descreve os diferentes estilos de interface assíncrona voltada para sistemas GALS. Fig. 1 mostra uma interface genérica com o módulo síncrono. Sistemas GALS têm sido usados com sucesso em várias implementações

ASIC (*Application Specific Integrated Circuit*) [13] e em FPGA [14].

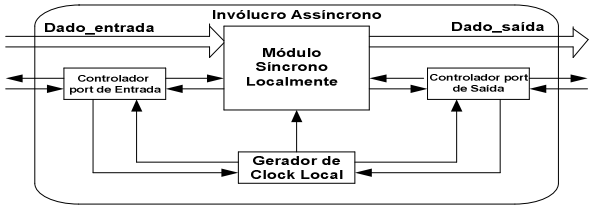

Fig.1. Invólucro assíncrono.

Interfaces assíncronas que usam *ports* de comunicação são interessantes porque eles permitem remover o protocolo (esquema) *handshake* assíncrono do módulo síncrono, assim o módulo síncrono pode ser projetado com técnicas usuais (*standard*) do projeto síncrono. Este estilo viabiliza fortemente o uso de módulos IP's síncronos.

Apesar do sistema GALS, ter solucionado os problemas do sinal de *clock* global, a comunicação entre os módulos é realizada no paradigma assíncrono, portanto sujeito aos problemas intrínsecos deste paradigma. Surgem diversos obstáculos para os projetistas do paradigma síncrono na síntese dos controladores de comunicação (*ports*):

1. A especificação mais familiar (*extended burst-mode* – XBM) descreve somente alguns tipos de ports [15];
2. Itens já citados como escassez de ferramentas disponíveis para síntese automática e a lógica assíncrona não é familiar;
3. Necessidade de resolver problemas de hazard em alguns circuitos principalmente na plataforma FPGA [16].

Este artigo apresenta um método para síntese de controladores *ports* (controladores assíncronos voltados para comunicação em GALS). O método parte da descrição dos *ports* no grafo multi rajada (GMR) [17]. A especificação GMR é familiar aos projetistas do paradigma síncrono, porque ela é baseada em diagrama de estados. Ela permite uma descrição mais compacta, quando comparada com especificações mais populares [15,18]. O método não precisa do conhecimento da teoria de síntese assíncrona. Ele sintetiza no estilo por mapeamento direto, onde cada estado está associado um elemento de controle (memória) e pode ser facilmente realizado manualmente. As vantagens do método são: *a)* a possibilidade de obter circuitos menores, quando comparado com o método por mapeamento direto de [19]; *b)* procedimento mais simples que pode ser manual, quando comparado com o método por mapeamento direto de [20].

II. ESPECIFICAÇÃO DE CONTROLADORES PARA GALS

Sistemas GALS requerem esquemas de comunicação assíncrona. Comunicação assíncrona pode usar protocolo de comunicação *handshake* de duas fases ou de quatro fases [8]. Os *ports* podem atuar na forma ativa (gerando o sinal de pedido ─ *request*) ou passiva (gerando o sinal de aceito ─ *acknowledge*). No projeto GALS há dois tipos de controlador de comunicação [21,22]: *a) port* de demanda (*demand*); *b) port* de pesquisa (*poll*). No *port* de demanda, o dado que está sendo transferido é imediatamente requerido depois da comunicação. Portanto, neste tipo de controlador, o relógio deve imediatamente ser interrompido, e reativado quando a comunicação foi realizada. No *port* de pesquisa, o relógio não é interrompido imediatamente. Ele é que decide quando está seguro enviar os dados. O relógio é interrompido somente em casos em que há necessidade de tempo adicional, para resolver problemas relacionados com metaestabilidade.

Especificações de controladores assíncronos

Duas importantes especificações para descrever controladores *ports* são: grafo de transição de sinais (*Signal Transition Graph* ─ STG) e modo rajada (Burst-Mode ─ BM):

─ *STG*: proposta por Chu em seu PhD [18], descreve controladores assíncronos que se comunicam com o ambiente externo no modo entrada/saída (E/S). No modo E/S a ocorrência de um evento na saída, imediatamente permite a ocorrência de um evento na entrada. O STG é baseado em *petri-net* e descreve diferentes classes de controladores assíncronos. O forte do STG é a capacidade em descrever todos os tipos de concorrência e seqüencia de eventos. A fraqueza é que nem todas as descrições são implementáveis. Elas não são populares aos projetistas do paradigma síncrono e podem se tornar confusas [8].

─ *BM:* formalizada por Nowick [23] e estendida por Yun (XBM) [15] e por Oliveira et al. (GMR) [17]. Ela é usada para descrever máquinas de estado finito assíncrono modelo Mealy. Estas máquinas interagem com o ambiente no modo fundamental generalizada. Neste modo, um novo conjunto de sinais de entrada (entrada rajada) só será ativado se o controlador estiver em um estado estável. Estas máquinas obedecem ao modelo de atraso *bounded gate and wire delay*, isto é, elas funcionam corretamente neste modelo de atraso. O forte do BM e extensões é que todas as descrições válidas são implementáveis. Elas são familiares, porque são baseadas em grafo de estados e não em eventos e são mais compactas que o STG. A fraqueza é que a especificação BM é pobre em descrever concorrência entre entradas e saídas e seqüência. A especificação XBM (*extended burst-mode*) permite descrever somente uma limitada concorrência, mas é pobre em seqüência. Em [17], a especificação GMR (grafo multi rajada), que é uma extensão da especificação XBM, permite descrever tipos de concorrência e seqüência.

III. ESPECIFICAÇÃO GRAFO MULTI RAJADA

A especificação BM pertence a uma classe de especificações que permitem múltiplas mudanças de entrada. Ela é representada por um grafo em que os vértices representam estados estáveis, enquanto os arcos representam as transições de estado, que são rotulados por uma entrada rajada (*burst*) e saída rajada (*burst)* (podendo ser vazia) [23]. Uma entrada rajada é formada por um conjunto de sinais de entrada que podem ser ativadas em qualquer ordem e em qualquer tempo. Yun [15] propôs uma extensão á especificação BM (*extended burst-mode* – XBM) adicionando duas características: sinais irrelevantes (*direct don't-care*) permitem que um sinal de entrada possa mudar simultaneamente com um sinal de saída, e sinais sensíveis ao nível, com comportamento não-monotônico que possam ser usados como sinais condicionais e são importantes em

ambientes heterogêneos. O XBM deve satisfazer as restrições apresentadas em [15].

Para descrever um limitado comportamento concorrente entre os sinais de entrada e de saída, Oliveira et al. [17,24] propôs uma nova especificação chamada grafo multi rajada (*Multi-Burst Graph*). A especificação GMR é uma extensão da XBM. Como no XBM, GMR é representada por um grafo, onde cada vértice representa um estado estável e cada arco representa uma transição. Cada transição no GMR pode ser ativada por: *1)* uma entrada rajada; ou *2)* uma expressão rajada. A especificação GMR introduz três operadores: (OR) →causalidade OR entre duas entradas rajadas; (CO) → transição concorrente entre rajadas; e (SEQ) → transição seqüência de rajadas. As expressões rajadas são baseadas nestes operadores, com isso aumenta as possibilidades de descrever concorrência Entrada/Saída e seqüência de eventos. A transição de estado que contem a expressão rajada deve satisfazer a propriedade de codificação de estado único (CEU) [18]. A propriedade CEU mais as restrições que a especificação GMR deve obedecer garantem as condições de implementabilidade [17,24].

Fig. 2 mostra uma especificação GMR e o estado inicial é *0*. As entradas são: *a, b, c, d*. Onde *a* é um sinal do tipo nível e os outros sinais são do tipo transição. As saídas são: *x, y, z*. O sinal terminando com (**+**) significa ativação 0→1 e terminando com (**–**) significa ativação 1→0. Na transição de estado *5 → 6* está presente o operador OR. Ele permite a causalidade OR entre os sinais *b* e *d*. Na transição seguinte *6→7*, o sinal *b~* tem um valor indeterminado e o sinal *d+* é *d=1* ou *d = 0→1*. Na transição de estado *2 →3*, o sinal *d** é irrelevante. O operador CO está presente nas transições de estado *0→3* e *4 →0*. O operador SEQ (>) está presente nas transições de estado *1→5* e *4→0*. O comportamento dos operadores CO e SEQ são detalhados em [17,24].

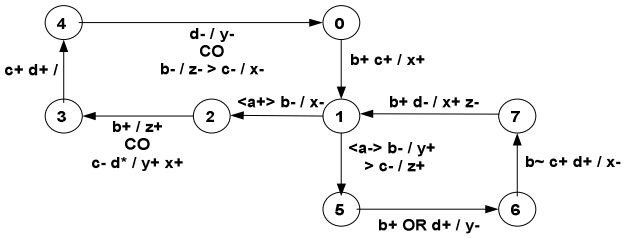

Fig. 2. Especificação GMR.

Transformação STG em GMR

Fig. 3,4 mostram respectivamente os controladores *ports* de entrada e saída descritos em STG de [25]. Fig. 5,6 mostram estes *ports* descritos em GMR. Usando principalmente o conceito de expressão rajada com o operador seqüência (>) os controladores *ports* descritos em STG podem ser descritos em GMR. A conversão é realizada por aplicar os disparos dos *tokens* no STG, que é capturada por uma entrada rajada/saída rajada ou por uma expressão rajada. Para definir a hierarquia na expressão rajada usam-se parênteses ou rótulos com índices. Fig. 5 mostra a transição de estado 0→1 rotulada com a expressão rajada En+ / Stoph+ (Req+ > Ack+/ Req-), onde o sinal Stoph é ativado concorrentemente com o termo (Req+ > Ack+/Req-)

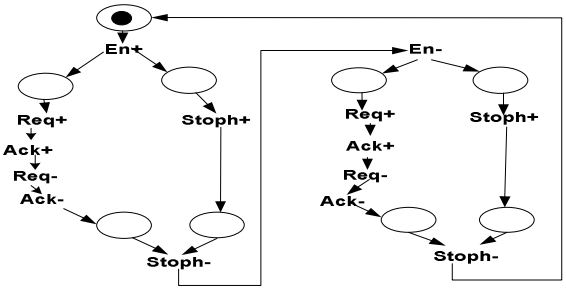

Fig. 3. Especificação STG: *port* de entrada de [25].

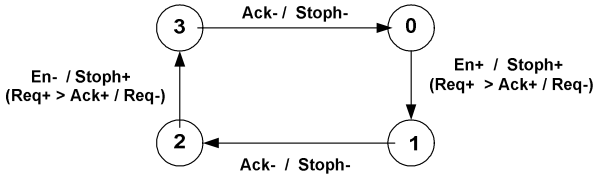

Fig. 5. Especificação GMR: *port* de entrada de [25].

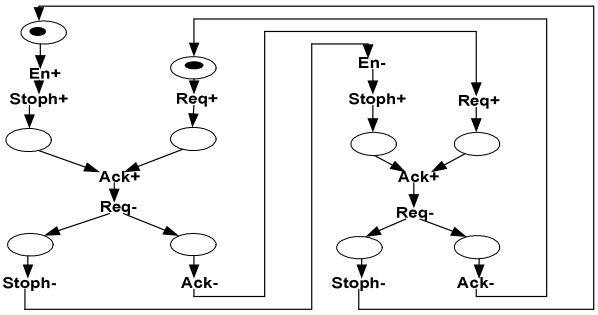

Fig. 4. Especificação STG: *port* de saída de [25].

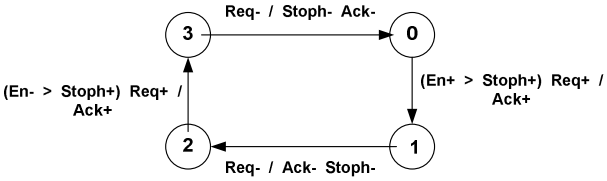

Fig. 6. Especificação GMR: *port* de saída de [25].

Transformação XBM em GMR

Fig. 7,8 mostram respectivamente os controladores *ports* de entrada e saída descritos em XBM e BM de [21]. Fig. 9,10 mostram estes *ports* descritos em GMR. Duas ou mais transições de estado consecutivas no XBM ou BM, onde os estados contêm somente uma transição que chega e que sai, podem ser descritas por uma expressão rajada que envolve o operador sequencia. Fig. 9 mostra a transição de estado 0→1 rotulada com a expressão rajada $En_{\_D}+ R_R+ / R_{\_CLK}+ > A_{\_CLK}+ / A_R+$ onde $A_{\_CLK}+$ é ativado imediatamente após a ativação de $R_{\_CLK}+$.

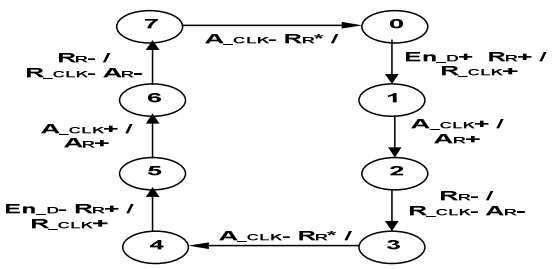

Fig. 7. Especificação XBM: *port* D_tipo entrada de [21].

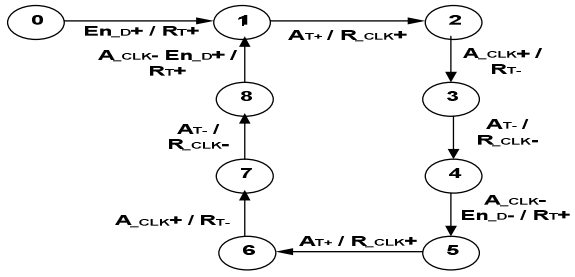

Fig. 8. Especificação BM: *port* D_tipo saida de [21].

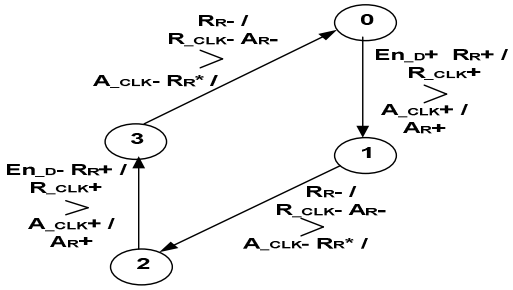

Fig. 9. Especificação GMR: *port* D_tipo entrada de [21].

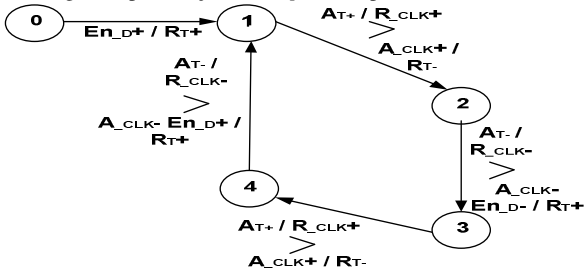

Fig. 10. Especificação GMR*: port* D_tipo saída de [21].

IV. MAPEAMENTO DIRETO DE CONTROLADORES ASSÍNCRONOS

O método usa o estilo de mapeamento direto para síntese de controladores assíncronos GMR proposto em [24], onde cada estado do GMR é assinalado um elemento de memória (controle). Este artigo apresenta uma extensão para o método de [24] que é o uso do latch RS como elemento de controle. O elemento de controle tem os sinais de entrada [*Ri, Ai*] e os sinais de saída [*Ro, Ao*]. O sinal de entrada *Ri* habilita o estado presente. O sinal de saída *Ro* inicia a ativação do próximo estado. O sinal de saída *Ao* inicia o processo de desabilitar do estado anterior. O sinal de entrada *Ai* desabilita o estado. A função do elemento de controle é armazenar o estado presente e habilitar, desabilitar a seqüência de estados que estão sendo processados. Fig. 11a,b mostram o latch RS rotulado.

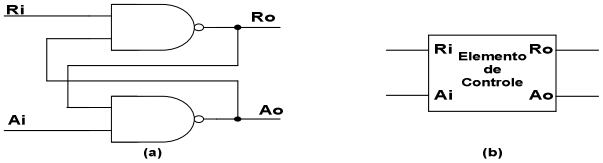

Fig.11. Elemento de controle: a) latch RS; b) símbolo.

Metodologia de Síntese

O comportamento do *port* de comunicação do sistema GALS é capturado inicialmente pela especificação GMR, como ilustrado na Fig. 5. Fig. 12 mostra a estrutura lógica da arquitetura alvo para os estados (i,j) dos controladores *ports*, compostos pelos blocos LC, EC e LO.

O procedimento de síntese do método MAP_DIR_GMR consiste de sete passos [24]:

1. Cada estado da especificação GMR será representado por um EC. Caminhos com duas transições de estado devem ter um EC auxiliar.
2. Para cada EC, há um bloco LC (lógica de condição ─ equação de excitação). A saída do LC é conectada no *Ri* do EC. ECs auxiliares não precisam do bloco LC.
3. Faça as conexões *Ro-i→LC-j* para cada transição de estado, onde o sinal *Ro* é da célula do estado inicial e o bloco *LC* está relacionado com o estado final.
4. Faça as conexões *Ao-j→Ai-i* para cada transição de estado na direção oposta. Quando houver duas ou mais conexões que chegam ao sinal *Ai* (estado de decisão) gera-se uma junção (*JOIN*).
5. Para cada estado *j* do GMR que corresponde ao bloco LC_J , extrair a função booleana do tipo soma de produto. Cada transição de estado que sai do estado *j* gera um produto com os sinais de gatilho pertencentes á entrada rajada ou expressão rajada.
6. Para cada junção (JOIN) de conexões, substituir por uma porta OR.
7. Usando sinais *Ro* dos ECs e sinais de entrada que atuam com o operador (>), extrair as funções Booleanas minimizadas do tipo soma de produtos dos sinais de saída ou funções F_{SET} e F_{RESET} quando envolve o latch RS.

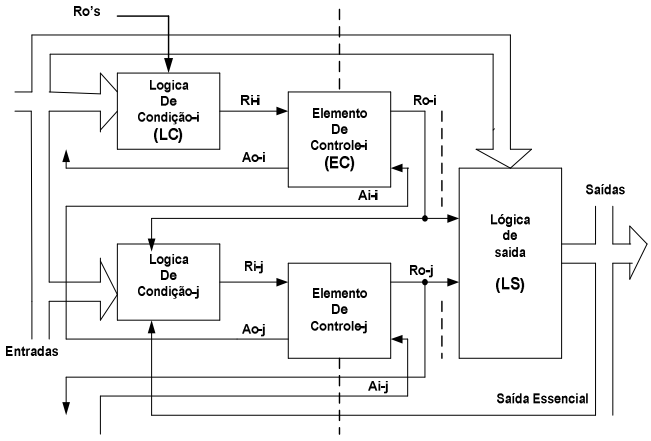

Fig. 12. Arquitetura alvo para GMR.

V. EXEMPLO

As Fig. 13 e 14, respectivamente, mostram as especificações GMR dos *ports* de comunicação de entrada e saída que foram propostos em [26]. Eles foram descritos originalmente na especificação STG. Para ilustrar o nosso método, escolhemos o *port* de entrada da Fig. 13. Fig. 15 mostra o passo 1, que é a substituição de cada estado do GMR por um EC. Fig. 16 mostra os passos 2 e 3, que realizam as conexões dos blocos lógicos (LC) com o respectivo EC, e as conexões de ativar estado (Ro). Fig. 17 mostra o passo 4 que realiza as conexões de desativar estado *Ao→Ai*. Fig. 18 mostra o passo 5, que é a obtenção das equações Booleanas de cada LC. Cada produto de uma LC

corresponde os sinais de entrada rotulados na transição de estado mais o sinal *Ro* do estado inicial. O passo 6 não foi necessário. Fig. 19 mostra o passo 7 que foi obter as equações Booleanas dos sinais de saída, onde o sinal *ap* usou o lath RS e o sinal *g* foi soma de produto.

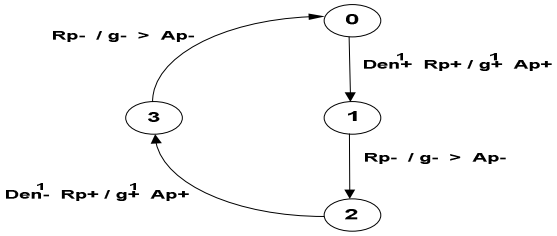

Fig. 13. Especificação GMR: port de entrada

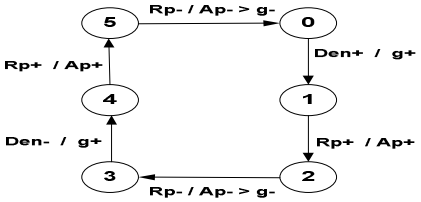

Fig. 14. Especificação GMR: port de saída.

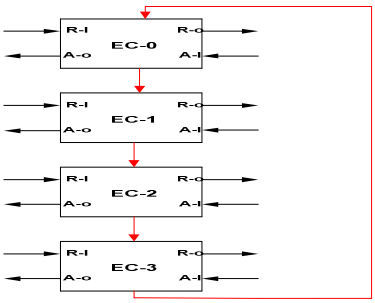

Fig. 15. Rede inicial de células de controle.

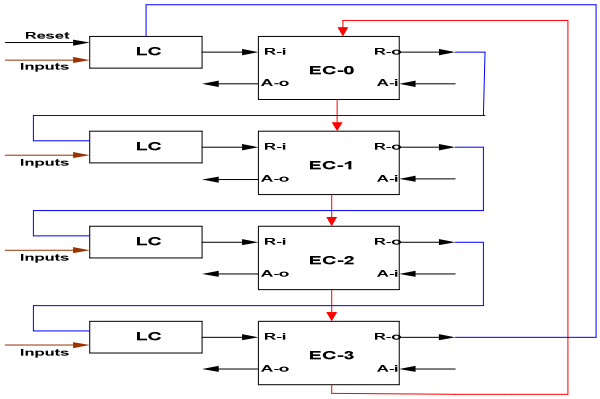

Fig. 16. Rede de células de controle: conexões.

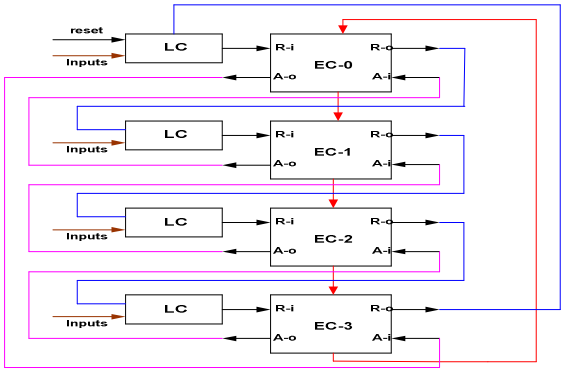

Fig. 17. Rede: conexões finais.

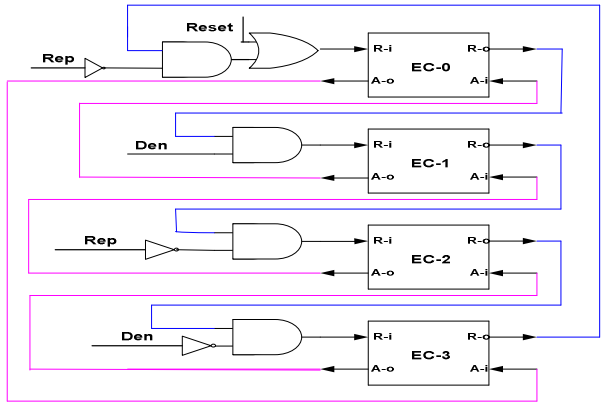

Fig. 18. Rede: funções Booleanas (LC).

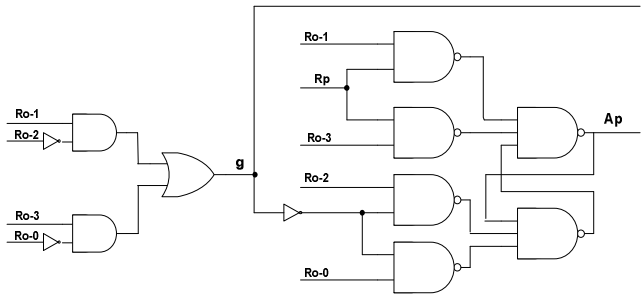

Fig. 19. Port de saída: lógica de saída

VI. DISCUSSÃO & RESULTADOS

As grandes vantagens do sistema GALS são: *a)* a possibilidade de reusar *IPs* existentes; *b)* os *IPs* operam na freqüência original; *c)* uso de ferramentas síncronas comerciais para projeto e verificação de novos *IPs*; *d)* redução drástica no esforço da análise de temporização; e) redução da interferência eletromagnética; *f)* potencial na redução do consumo de energia [28]; *g)* eliminação do problema de *clock skew* [29]; e *h)* controlador *port* robusto [30]. A habilidade de GALS-VLSI processando com *IPs* em diferentes freqüências e tensões de alimentação, também contribuem para a redução da potência. Todavia, os sistemas GALS têm os seus próprios inconvenientes, por exemplo, metaestabilidade e *overhead* de comunicação [12].

Um problema no projeto GALS é a síntese dos controladores *ports*, principalmente quando envolve FPGAs. Os *ports* da interface assíncrona SCAFFI [16] foram sintetizados na ferramenta MINIMALIST [31]. O *port* de entrada tem risco essencial do tipo transiente [7,8]. A solução sugerida pelos autores para este *port* foi usar um difícil mapeamento baseado em *hard* macros. O método MAP_DIR_GMR sintetiza o *port* SCAFFI de entrada, e usa um procedimento mais simples para solucionar o problema de risco essencial [30]. O *port* SCAFFI de saída foi descrito em VHDL estrutural usando o método MAP_DIR_GMR e implementado no dispositivo EP2515F48C3 da família Stratix II, da ALTERA [32]. A Simulação deste *port* está na Fig. 20, mostrando que funcionou corretamente, portanto não necessitou da inserção de elementos de atraso ou escolha de LUTS para este circuito.

A grande vantagem do método MAP_DIR_GMR é permitir que a especificação STG seja sintetizada manualmente por mapeamento direto, com uma otimização sub-ótima, inicialmente, através da conversão do STG em GMR. O método por mapeamento direto voltado para STG que foi proposto por Sokolov, et al. [20] é muito complexo para se aplicar manualmente, apesar de potencialmente se obter uma melhor otimização, ainda que não comprovada. As Fig. 21 e 22 mostram o potencial de descrição do GMR de 16 controladores *ports* e a sua capacidade de compactação. Para os ports em XBM e BM, houve uma redução de 28% no número de estados e transições de estado. Para os *ports* em STG, houve uma redução de 34% nas transições. Estas reduções simplificam a síntese e uma otimização sub-ótima é obtida no circuito final.

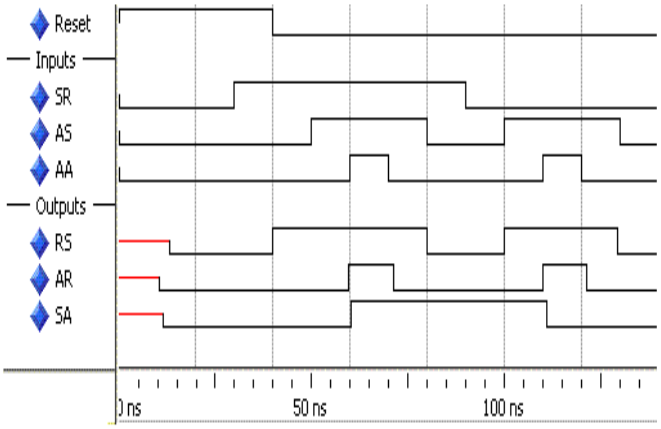

Fig. 20. Simulação: port saída SCAFFI de [16].

	Especificação			Espec. GMR
	Espec. Estilo	N_en / N_sai	N_estados/ N_trans.	N_estados/ N_trans.
Port Passivo entrada [11]	XBM	3 / 3	4 / 4	4 / 4
Port Ativo Entrada [11]	XBM	3 / 3	4 / 4	4 / 4
Port Passivo Saída [11]	XBM	3 / 3	4 / 4	4 / 4
Port Ativo Saída [11]	XBM	3 / 3	4 / 4	4 / 4
Port D_tipo Entrada [21]	XBM	3 / 2	8 /8	4 / 4
Port P_tipo Entrada [21]	XBM	3 / 2	8 / 8	4 / 4
Port Entrada SCAFFI [16]	BM	3 /3	10 / 10	7 / 7
Port Saída SCAFFI [16]	BM	3 / 3	10 / 10	7 / 7

Fig. 21. Especificações: conversão (XBM,BM)→GMR.

	Espec. STG		Espec. GMR
	N_en / N_sai	N_trans.	N_estados/ N_trans.
Port_W [27]	2 / 2	8	3 / 3
Port_R [27]	2 / 2	7	3 / 3
Port entrada [26]	2 / 2	17	4 / 4
Port Saida [26]	2 / 2	14	6 / 6
Port Entrada [25]	3 / 2	14	4 / 4
Port saida [25]	3 / 2	14	4 / 4

Fig. 22. Especificações: conversão STG→GMR.

VII. CONCLUSÃO

A implementação de sistemas GALS nas plataformas VLSI ou FPGA pode ser útil no projeto de sistemas embarcados complexos, seja porque há potencial de redução de potência, como também a redução dos problemas relacionados com o *clock* global. Um dos principais problemas do projeto GALS está na interface, onde os controladores *ports* merecem uma atenção especial. Neste artigo, foi apresentado um método que sintetiza controladores *ports* de forma sistematizada, podendo ser realizado manualmente. O método apresentado sintetiza eficientemente controladores *ports* que estão descritos ou em STG ou em XBM/BM que são as duas especificações mais populares. O método parte da especificação GMR que descreve facilmente especificações STG e XBM/BM e que permite uma síntese por mapeamento direto fácil. Como trabalho futuro pretende-se aplicar este método no projeto GALS de um radio definido por software, o que é importante no setor aeroespacial.

REFERÊNCIAS

[1] G. DE Micheli, "*An Outlook on Design Technologies for Future Integrated Systems*," IEEE Trans. on CAD of Integrated Circuits and Systems, vol28, no.6, pp. 777-789, June 2009.

[2] D. Sylvester and K. Keutzer, "Getting to the Bottom of Deep Submicron,' Proc. ICCAD, pp. 203-211, November 1998.

[3] R. ho, K. W. Mal and M. A. Horowitz, "*The Future of Wires*,' Proc. of the IEEE, vol. 89, no. 4, pp. 490-504, April 2001.

[4] E. G. Friedman, "*Clock Distribution Networks in Synchronous Digital Integrated Circuits,*" Proc. of. The IEEE, vo. 89, pp. 665-692, 2001.

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

[6] 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.

[7] A. J. Martin and M. Nystrom, "*Asynchronous Techniques for System-on-Chip Design,"* Proc. of the IEEE, vol.94, no. 6, pp.1089-1120, June, 2006.

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

[9] D. M. Chapiro, *Globally-Asynchronous Locally-Synchronous Systems*, PhD thesis, Stanford University, October 1984.

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

[11] D. S. Bormann and P. Y. K., "Asynchronous Wrappers for Heterogeneous Systems," Proc. Int. Conf. Computer Design (ICCD), pp.307-314, October 1997.

[12] P. Techan, M. Greenstreet, and G. Lemieux, "*A Survey and Taxonomy of GALS Design Styles,*" IEEE Design & Test of Computers, vol. 24, pp.418-428, September-October 2007.

[13] F. K. Gurkaynak, et al., "GALS at ETH Zurich: Success or Failure ?," , Proc. 12th IEEE Int. Symposium on Asynchronous Circuits and Systems, pp. 150-159, 2006.

[14] A. Kumala, et al., "Reliable GALS Implementation of MPEG-4 Encoder with Mixed Clock FIFO on Standard FPGA," Int. Conf. on Field Programmable Logic and Application, pp. 1-6, 2006.

[15] K. Y. Yun and D. L. Dill, "*Automatic Synthesis of Extended Burst-Mode Circuits: Part I (Specification and Hazard-Free Implementation) and Part II (Automatic Synthesis)*," IEEE Trans. on CAD of Integrated Circuit and Systems, vol. 18:2, pp. 101-132, February 1999.

[16] J. Pontes, et al., "SCAFFI: an Intrachip FPGA asynchronous interface based on hard macros," 25th Int. Conf. on Computer Design, pp.541-546, 2007.

[17] D. L. Oliveira, et al., "Miriã: a CAD Tool Synthesize Multi-Burst Controllers for Heterogeneous System," *Microelectronics Reliability*, vol. 43, pp. 201-215, 2003.

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

[19] D. L. Oliveira, et al., “Synthesis by Direct Mapping of Asynchronous Controllers from Extended Burst-Mode Specification,” XVI Workshop Iberchip, Foz de Iguaçu – Brazil, 2010.
[20] D. Sokolov, A. Bystrov and A. Yakovlev, ”Direct mapping of low-latency asynchronous controllers from STGs,” *IEEE Trans. CAD of Integration Circuits and Systems*, vol. 26, no. 6, June 2007.
[21] J. Muttersbach, “Globally-Asynchronous Locally-Synchronous Architectures for VLSI Systems,” Ph.D. Thesis, ETH, Zurich, 2001.
[22] A. Reddy Ravi, “Globally-Asynchronous, Locally-Synchronous Wrapper Configurations for Point-to-Point and Multi-Point Data Communication,” Master of Science, University of Central Florida, 2004.
[23] S. M. Nowick, Automatic Synthesis of Burst-Mode Asynchronous Controller, PhD thesis, Stanford University, 1993.
[24] D. L. Oliveira e E. Lussari, “Synthesis by Direct Mapping of AFSMs from Flexible Multi-Burst Graph Specification,” XVII Workshop Iberchip, Bogotá, Colombia, 2011.
[25] J. Carlsson, *Studies on Asynchronous Communication Ports for GALS Systems,* Lic. Doctorate Thesis, Linkoping Studies in Science and Technology, Linkoping University, Sweden, June 2005.
[26] E. Amini, M. Najibi and H. Pedram, ”Globally Asynchronous Locally Synchronous Wrapper Circuit based on Clock Gating,” Proc. Emerging VLSI Technologies and Architectures, pp.193-199, 2006.
[27] S. Zhuang, et al., “An Asynchronous Wrapper with Novel Handshake Circuits for GALS Systems,” Proc. IEEE, Communications, Circuits and Systems and West Sino Expositions, pp.1521-1525, 2002.
[28] A. Hemani, et al., “Lowering power consumption in clock by using Globally Asynchronous Locally Synchronous design style,” Proc. IEEE 36th Conf. DAC, pp.873-878, 1999.
[29] X. Jia and R. Vemuri, “Using GALS Architecture to Reduce the Impact of Long Wire Delay on FPGA Performance,” Proc. IEEE ASP-DAC, pp. 1260-1263, 2005.
[30] D. L. Oliveira e E. Lussari, “Synthesis of Robust Controllers for GALS-FPGA from Multi-Burst Graph Specification,” Proc. IEEE VII Southern Conference on Programmable Logic- SPL, Córdoba, 2011.
[31] R. Fuhrer, et al., “Minimalist: An environment for the synthesis, verification and testability of burst-mode asynchronous machines,” Columbia Univ., NY, Tech. Rep. TR-CUCS-020-99, July 1999.
[32] Altera Corporation, 2013, www.altera.com.