

# Métodos para Desenvolvimento e Distribuição de IP Cores

José Carlos Palma, Fernando Moraes, Ney Calazans

Faculdade de Informática – PUCRS  
Av. Ipiranga, 6681 – Porto Alegre – CEP: 90619-900  
{jpalma, Moraes, Calazans}@inf.pucrs.br

**Abstract:** *This work addresses the need of using predesigned and preverified circuit designs (cores) to implement complex systems. The design flow using cores to implement system-on-a-chip devices is discussed. A set of tools to implement a core distribution system is presented. The implemented environment allows to share soft and hard cores. Hard cores, distributed as FPGA bitstreams, can be useful only if partial reconfiguration is supported. We first address complete reconfiguration, presenting a system that allows remote and complete bitstream reconfiguration. The last part of this paper proposes an approach that virtualizes the hardware, using dynamic and partial reconfiguration.*

**Keywords:** *reconfiguration, FPGA, cores.*

## 1. Introdução

O avanço tecnológico na construção de sistemas digitais complexos é tal que permite dispor hoje em um único circuito integrado cerca de 25 milhões de transistores (microprocessadores) [1]. O ritmo desses avanços da tecnologia de fabricação tem se mantido exponencial nas últimas décadas, como atesta a *Lei de Moore* [2]. Esta lei é devida a Gordon E. Moore, que em 1965 observou que a densidade de componentes em circuitos integrados dobrava a intervalos regulares, inferindo que este comportamento perduraria por muito tempo ainda. O intervalo medido por Moore para que a densidade média de CIs dobrasse foi de 18 meses, o que ainda hoje permanece uma taxa estável [3].

A elevada complexidade dos projetos de circuitos integrados, acabou gerando duas consequências principais:

- a) É preciso que se trabalhe em níveis mais abstratos de projeto, pois o uso de técnicas como captura de esquemáticos tornou-se por demais complexa. Níveis mais abstratos de projeto são alcançados através da utilização de HDLs, como por exemplo VHDL [4] ou Verilog [5], permitindo ao projetista especificar seu sistema com um menor número de detalhes.
- b) Tornou-se necessário o reuso de módulos pré-caracterizados para o desenvolvimento de novas aplicações. O uso de módulos pré-caracterizados (*cores*) é uma forma de reduzir a complexidade dos atuais sistemas digitais. Assim, o sistema digital passa a constituir-se de poucos *cores*, os quais implementam funções complexas, tais como: *Fast Ethernet Mac*, *PCI bus*, *Utopia Master*.

Este artigo é organizado da seguinte forma. Na Seção 2 é feita uma revisão das características e das dificuldades no uso de *cores*. As Seções 3 e 4 apresentam respectivamente uma proposta de ambiente para distribuição de *soft* e *hard cores*. A Seção 5 apresenta a proposta de um método para conexão entre *cores* em funcionamento em um mesmo FPGA, de forma que um *core* possa ser substituído por outro. Finalmente, a Seção 6 apresenta as conclusões deste trabalho e os trabalhos futuros.

## 2. Cores

Um *core* é nada mais do que uma descrição de um sistema digital. Estes sistemas digitais são pré-projetados, pré-verificados, e geralmente de diversos graus de complexidade. Os *cores* podem ser usados para construir aplicações maiores e mais complexas em um circuito integrado.

Freqüentemente, estes componentes são produtos de tecnologia, *software*, e conhecimento que são submetidos à patentes e direitos de propriedade. Em outras palavras, um *core* representa uma propriedade intelectual que o desenvolvedor autoriza para o usuário do mesmo [6].

Quanto à sua disponibilização, pode-se classificar um projeto de *core* como sendo licenciável ou proprietário. Um *core* licenciável é aquele em que o fornecedor licencia um projeto e um conjunto de ferramentas conferindo a tarefa de projeto do sistema e sua fabricação ao cliente (comprador). Um *core* proprietário é aquele em que a preocupação principal do fornecedor é proteger sua propriedade intelectual.

Quanto à flexibilidade os *cores* podem ser classificados em três categorias: *hard*, *firm* e *soft*. A Tabela 1 ilustra a classificação dos *cores* conforme cinco critérios.

**Tabela 1 – Três classificações de IP cores baseado em cinco critérios.**

|  | <b>Hard core</b>   | <b>Firm core</b>  | <b>Soft core</b>  |
|--|--|---|---|
| <b>Rigidez</b>                             | A organização é predefinida.                                   | Combinação de código fonte e netlist dependente de tecnologia.                            | Apresenta um código fonte comportamental independente da tecnologia.          |
| <b>Modelagem</b>                           | Modelado como um elemento de biblioteca.                       | Combinação de blocos sintetizáveis fixos. Permite compartilhar recursos com outros cores. | Sintetizável com diversas tecnologias.  |
| <b>Flexibilidade</b>                       | Não pode ser modificado pelo projetista.                       | A personalização de funções específicas é dependente da tecnologia.                       | O projeto pode ser modificado e independe da tecnologia.                      |
| <b>Previsibilidade</b>                     | Temporização é garantida.                                      | Caminhos críticos com temporizações fixas.  | A temporização não é garantida, podendo não atender os requisitos do projeto. |
| <b>Proteção da propriedade intelectual</b> | Alta. A descrição normalmente corresponde a um <i>layout</i> . | Média   | Muito pequena. Código fonte aberto.   |

Em 1997, cerca de 40% das partes dos sistemas digitais eram compostos por módulos reutilizáveis, devido ao grande aumento de produtividade obtido através do uso destes. Existem previsões indicando que em 2012 esta porção aumente para 90% [1]. Este fato justifica o desenvolvimento de técnicas para o desenvolvimento e distribuição de *cores*,

### 2.1 Distribuição de Cores na Internet

Existem hoje na Internet centenas de web sites de fornecedores de *cores* abertos ou com propriedade intelectual protegida. A maioria dos web sites que distribuem *cores* na Internet são de empresas comerciais, que desenvolvem seus próprios módulos e os vendem através da rede. A Tabela 2 apresenta alguns exemplos de *sites* que distribuem *cores*.

Atualmente, algumas destas empresas estão investindo em sistemas de *test drive* de *cores*, ou seja, o cliente pode simulá-los através de simuladores HDL padrão antes de comprá-los. O cliente entra em contato com o fornecedor, recebe um *link* para FTP via e-mail e pode fazer o download do *soft core* criptografado, de um modelo de simulação também criptografado, de padrões de teste e de uma documentação abreviada do produto.

Outra categoria de distribuidores de *cores* encontrada na Internet é a de entidades que promovem a distribuição gratuita de módulos com propriedade intelectual livre, também

chamados de *cores* abertos. Devido ao fato dos *cores* pagos serem bastante caros, a quantidade dessas entidade vem crescendo cada vez mais. Estas entidades geralmente disponibilizam *soft cores* sintetizáveis já simulados, depurados e testados, que podem ser capturados em seus web sites.

**Tabela 2 – Distribuição de cores na Internet.**

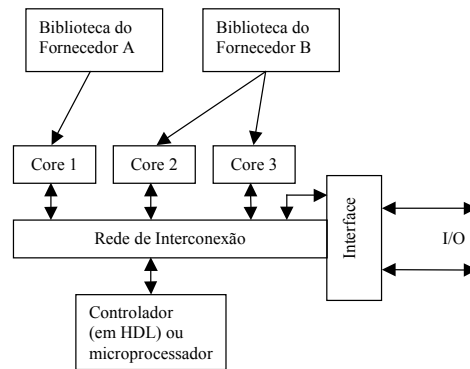
| Fornecedor                 | Descrição      | Principais aplicações  | Endereço  | IP         |
|----------------------------|----------------|--|---|------------|
| Free-IP                    | VHDL           | RAMs parametrizáveis, RISC8, DES   | <a href="http://www.free-ip.com/">http://www.free-ip.com/</a>   | Aberto     |
| Opencollector              | VHDL e Verilog | Bibliotecas, Sistemas Embarcados, Processadores                            | <a href="http://www.opencollector.org">http://www.opencollector.org</a>   | Aberto     |
| In-Silicon                 | Verilog        | Core PCI 33MHz, Controlador USB, MAC Ethernet                              | <a href="http://www.in-silicon.com">http://www.in-silicon.com</a>   | Teste/Pago |
| Vautomation                | VHDL           | Microprocessadores, USB, Ethernet, Ferramentas de Projeto                  | <a href="http://www.vautomation.com">http://www.vautomation.com</a>   | Pago       |
| CMOSexod                   | VHDL e Verilog | Processador CISC, DSP 12-bit, Controlador SDRAM                            | <a href="http://www.cmosexod.com">http://www.cmosexod.com</a>   | Aberto     |
| FMF                        | VHDL e Verilog | Bibliotecas de FIFOS, de SRAMs e de Processadores                          | <a href="http://www.vhdl.org/fmf">http://www.vhdl.org/fmf</a>   | Aberto     |
| Design-Reuse               | VHDL e         | Microcontrolador de 8-bits, Controladores de SDRAM, Ferramentas de Projeto | <a href="http://www.design-reuse.com">http://www.design-reuse.com</a>   | Teste/Pago |
| Opencores                  | VHDL e Verilog | Microprocessadores, Memórias, DSPs, Controladores de Video                 | <a href="http://www.opencores.org">http://www.opencores.org</a>   | Aberto     |
| Arasan Chip Systems        | VHDL e Verilog | Core USB Genérico, Controlador de Hub USB 2.0                              | <a href="http://www.arasan.com/">http://www.arasan.com/</a>   | Pago       |
| Amphion Semiconduc-tor     | Netlist        | Processamento de Imagem, Processamento de voz, Codificação de Canais, DSP  | <a href="http://www.amphion.com">http://www.amphion.com</a>   | Pago       |
| Eureka Technology          | VHDL e Verilog | Controlador de sistema, Barramento PCI                                     | <a href="http://www.eurekatech.com">http://www.eurekatech.com</a>   | Pago       |
| Altera                     | VHDL e Verilog | Controladores de Memória, Comunicações, PCI, DSP                           | <a href="http://www.altera.com/products/ip/ipm-index.html">http://www.altera.com/products/ip/ipm-index.html</a> | Teste/Pago |
| Innovative Semiconduc-tors | -              | Codificadores/Decodificadores de Vídeo, USB                                | <a href="http://www.isi96.com">http://www.isi96.com</a>   | Pago       |

## 2.2 Sistemas Integrados em um Único Chip – SoCs

A realidade atual do mercado de projetos VLSI é caracterizada por *time-to-market* curto, alta complexidade e alto desempenho. Enquanto o *time-to-market* é o fator mais importante, complexidade e desempenho não podem ser comprometidos, sob o risco de se atingir o mercado com um produto não competitivo. Esta exigência está modificando a forma como circuitos VLSI são projetados.

O uso de *cores* permite ao projetista concentrar-se no sistema completo sem ter que se preocupar com o comportamento exato ou com o desempenho de componentes individuais. A Figura 1 [7] ilustra a arquitetura genérica de um SoC. Os *cores* são integrados através de uma rede de interconexão comercial ou adaptada, com um controlador e funções de interface com o meio externo. Eles podem ser novos ou herdados de projetos já existentes. Se eles forem obtidos de fontes diferentes, a integração e os testes podem ser difíceis, podendo até haver necessidade de que os *cores* sejam reprojitados, para adequá-los a um protocolo de interface comum.

Apesar de todas as vantagens inerentes à utilização de *cores*, identifica-se quatro grandes problemas que devem ser resolvidos para que se possa construir um SoC: (i) como integrar os *cores*; (ii) quais linguagens para descrição de sistemas utilizar; (iii) como proteger a propriedade intelectual do autor e do usuário do *core*; (iv) como testar os projetos de *cores*. Além disto, não existe quase nenhuma ferramenta de software que ajude o projetista a construir um SoC, integrando e configurando *cores* facilmente. A complexidade dos SoCs atuais, a falta de ferramentas de alto nível e as dificuldades acima fazem do reuso e *plug-and-play* soluções ainda inatingíveis.



**Figura 1 – Arquitetura SOC genérica.**

### 2.3 Fluxo de Projeto Usando Cores

O modelo tradicional de fluxo de projeto para SoCs é freqüentemente chamado de modelo cascata [8]. Neste modelo, a transição de fase para fase nunca retorna às atividades da fase anterior. Assim, o projeto é passado de uma equipe responsável por uma fase, para outra responsável pela fase seguinte, sem que haja muita interação entre elas.

Para grandes projetos, esta metodologia simplesmente não funciona. Grandes projetos têm um conteúdo de software suficiente para que haja a necessidade de se desenvolver o hardware e o software concorrentemente para assegurar a funcionalidade correta do sistema. Questões físicas do projeto devem ser consideradas no início do projeto para assegurar que o desempenho desejado seja também alcançado.

Na medida em que a complexidade aumenta, muitas equipes estão trocando o modelo cascata por um novo modelo, chamado de espiral. No modelo espiral, as equipes de projeto trabalham em múltiplos aspectos do projeto simultaneamente, melhorando cada etapa à medida que o projeto converge para sua conclusão.

### 3. Ambiente de distribuição de Soft Cores

A Seção anterior caracterizou os *cores*, mostrou a necessidade de utilizá-los para projetos de sistemas complexos e as dificuldades em sua utilização. Esta Seção apresenta a primeira contribuição deste trabalho, um ambiente desenvolvido para a distribuição de *cores*, independentemente de seu formato, via Internet. O objetivo em se criar tal ambiente é o de prover aos projetistas uma forma de distribuir seus projetos e de sistematizar a organização das informações contidas nos projetos. A interface principal do ambiente é mostrada na Figura 2.

Para que se manuseie adequadamente os projetos, é necessário ter uma estrutura de dados bem definida. Isto é ainda mais importante durante o desenvolvimento e a distribuição do *core*. A base de dados consiste em uma árvore de diretórios, onde ficam localizados todas as informações dos projetos [9].

Na interface apresentada na Figura 2 o projetista define os parâmetros principais do projeto (nome, responsável, descrição, pinagem externa) e tem acesso a diferentes janelas, onde são especificados os arquivos de documentação, fontes (em VHDL ou Verilog), teste e ferramentas. Em cada uma das janelas secundárias, além de se informar a localização dos arquivos, o usuário deve entrar com informações que descrevem cada um dos arquivos inseridos na base de dados.

Uma vez realizada a entrada de dados, procede-se à criação automática de um arquivo HTML com o nome do projeto e um diretório, contendo de forma hierárquica todos os

arquivos inseridos na base de dados do projeto. A página HTML contém *links* para os arquivos do projeto. O conjunto página HTML mais diretório de arquivos é inserido no servidor do *cores*, permitindo acesso remoto a todos que desejam utilizar o *core* desenvolvido.

| Pino | Orientação | Descrição |
|------|------------|-----------|
|      |            |           |
|      |            |           |
|      |            |           |
|      |            |           |
|      |            |           |

**Figura 2 – Interface do ambiente de distribuição de *soft cores*.**

Tal ferramenta de montagem de páginas para distribuição de *cores* encontra-se hoje operacional. O sistema está sendo melhorado à medida que novos *cores* são inseridos no servidor. Exemplo de trabalho em andamento é a inserção de controle de acesso (via autenticação de usuários), de forma a proteger a propriedade intelectual do desenvolvedor do *core*.

#### **4. Ambiente de reconfiguração e distribuição de Hard Cores**

A segunda contribuição deste trabalho é a reconfiguração e a distribuição de *hard cores* na forma de arquivos binários de configuração de dispositivos FPGAs (*bitstreams*).

Para realizar a distribuição de propriedade intelectual na forma de módulos já sintetizados (*hard cores*) para FPGAs é necessário que seja possível "encaixar" novos *cores* em um FPGA, sem interromper o funcionamento do sistema que está sendo executado no dispositivo configurável. Para que esta nova abordagem seja factível de ser implementada, é necessário preencher uma série de requisitos:

- i) FPGAs com arquitetura regular e disponibilidade de reconfiguração parcial. Alguns FPGAs, como por exemplo os da família Virtex, possuem arquitetura interna baseada em colunas, com endereçamento individual. Este fato permite que o dispositivo possa ter sua configuração modificada dinâmica e parcialmente.
- ii) Existência de ferramentas que permitam fixar a forma e a posição relativa dos *cores* no dispositivo programável. Exemplo de ferramenta que permite fixar a posição dos circuitos no interior do FPGA é o *Floorplanner* (Foundation - Xilinx).
- iii) Existência de ferramentas que permitam a geração de *bitstreams* parciais. O conjunto de classes JBITS permite teoricamente gerar um arquivo parcial, porém foi verificado pelos autores deste trabalho que o *bitstream* gerado é incorreto. Desta forma, não há ao conhecimento dos autores, ferramentas para geração e manipulação de arquivos parciais.
- iv) Existência de ferramentas que permitam o *download* de *bitstreams* parciais.
- v) Estrutura de conexão entre o *core* inserido e os demais *cores* já em operação no FPGA.

vi) Virtualização dos pinos de entrada e saída.

Os itens (iii) a (vi) carecem hoje de ferramentas. Não se encontra na literatura referência a ferramentas que manipulam arquivos parciais de configuração. O trabalho em desenvolvimento [10], não apresentado neste artigo, analisa ferramentas e dispositivos que tornam possível a reconfiguração parcial. O presente artigo apresenta a reconfiguração completa de *bitstreams*, assim como sua distribuição.

Como estudo de caso, é apresentado um reconfigurador total de *bitstreams*, que utiliza JBits. O JBits é constituído por um conjunto de classes Java que fornecem uma API (*Application Program Interface*) que permite manipular o *bitstream* da família de FPGAs Virtex [11]. Com base neste conjunto de classes, desenvolveu-se um ambiente para ser utilizado através da Internet, onde o usuário pode entrar com novos dados e alterar remotamente o conteúdo de um *bitstream*.

Um *bitstream* (*hard core*) é tido como uma caixa preta, cujo conteúdo não é visível nem possível de ser alterado. Porém, com o uso das classes do JBits, surge a possibilidade de reconfiguração de *hard cores*, mudando este conceito. De outro modo, seria necessário que se alterasse o código-fonte do *soft core* do circuito. Logo após, o mesmo precisaria ser novamente sintetizado (lógica e fisicamente).

A primeira etapa do trabalho consiste em gerar o *hard core*. Normalmente a modificação dos parâmetros que caracterizam o funcionamento de um dado circuito é feita através de uma interface com microprocessador externo ou chaves. Outra forma, muito mais simples, é através de dados escritos em blocos de RAM internos ao FPGA (por exemplo LUTRAM). Desta forma, ao gerarmos o *hard core*, todos os parâmetros deste são lidos de blocos de RAM. Estes blocos de RAM têm a posição física fixada dentro do FPGA através de restrições impostas no momento da síntese. O *core* assim sintetizado não tem nenhum parâmetro definido.

A segunda etapa do trabalho consiste em escrever uma aplicação em Java, utilizando as classes JBits, para acessar no arquivo de configuração os blocos de RAM que contêm os parâmetros que definem a funcionalidade do circuito.

Uma vez gerado o *bitstream* e a aplicação JAVA, procede-se à alteração do *bitstream* para configurá-lo conforme a necessidade do usuário. Após reconfigurar o *bitstream*, o usuário realiza o download do circuito através do mesmo ambiente.

É importante que se perceba o aspecto inovador desta abordagem. Uma vez gerado o circuito, pode-se modificá-lo sem a necessidade de re-síntese. Torna-se também extremamente simples modificar o comportamento do circuito, uma vez que toda a manipulação é feita em software. O software é baseado em um modelo cliente-servidor, rodando em um *browser* qualquer. Assim pode-se remotamente reconfigurar o hardware. Por exemplo, uma dada empresa pode reconfigurar o hardware de todos os seus clientes remotamente, sem necessidade de ir até o cliente, economizando assim custos de manutenção e atualização.

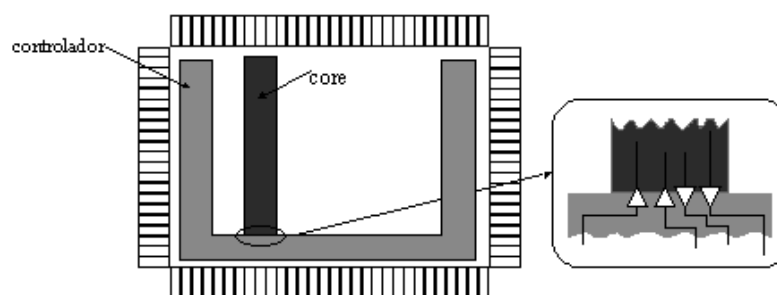
A aplicação para reconfiguração total e remota de *cores* é inserida também no ambiente de distribuição apresentado na Figura 2. É importante ressaltar que um *bitstream* completo só é utilizável no ambiente (placa) para o qual ele foi sintetizado, devido às restrições de pinos de entrada e saída. Para que um *hard core* seja distribuído e utilizado em diferentes ambientes, é necessária a reconfiguração parcial e a virtualização dos pinos de entrada e saída (conexão a um barramento padrão interno e não mais aos pinos de entrada e saída do FPGA).

## 5. Proposta de interface para conexão de *cores* em dispositivos Virtex

Como dito anteriormente, é possível alterar parcialmente a funcionalidade do hardware sem interromper o funcionamento do sistema. Para que este tipo de aplicação seja uma realidade em projetos de hardware é necessário definir uma estratégia de reconfiguração parcial dinâmica para os dispositivos FPGA. A idéia é que haja no FPGA uma estrutura análoga a uma interface PCI em um PC, onde se conectam dispositivos sem a modificação do sistema.

Até o presente momento não foi encontrada na literatura referência sobre uma proposta de interface intra-FPGA para que dispositivos reconfiguráveis possam receber *cores* de forma modular. Essa modularidade consiste em conectar e remover *cores* sem que haja necessidade de maiores alterações na lógica pré-existente no FPGA. Há técnica semelhante, voltada para o projeto de *cores* para ASICs. Como exemplo pode ser citado *Amba* [12] e o *CoreConnect* [13].

Um ponto de partida para a definição desta interface de comunicação entre diversos *cores* é estabelecer uma porção de hardware fixo (estático) no FPGA responsável pela comunicação com o mundo externo e um barramento para conexão dos *cores*, provido de pinos virtuais. E hardware estático é denominado controlador. A Figura 3 ilustra esta proposta.



**Figura 3 – Barramento de interconexão entre *cores*.**

O FPGA seria inicialmente carregado apenas com o controlador. O controlador comunica-se externamente com os pinos de entrada/saída do FPGA, e internamente com os sinais de entrada/saída do(s) *core*(s), através de pinos virtuais. Para os pinos virtuais será utilizado *buffers tristate*, presentes na arquitetura de FPGAs da família Virtex. Isto significa que os módulos de hardware a serem conectados, *cores*, somente comunicar-se-ão com o mundo externo através desta interface. Os *cores* serão carregados em tempo de execução, possuindo conexões aos sinais da interface. O controlador da interface será o árbitro deste barramento, gerenciando o aceite ou a rejeição de conexões, bem como o controle de conflitos relativos ao acesso aos pinos de entrada/saída do FPGA.

Em [14] é apresentado um sistema de reconfiguração utilizando como estudo-de-caso um sistema para processamento de imagens. Este sistema é reconfigurado quatro vezes para cada conjunto de dados (imagem) a ser processado. O FPGA primeiramente armazena o sinal de vídeo em uma memória, aplicando em seguida duas transformações diferentes sobre as imagens, e finalmente torna-se um modem e transmite o resultado do processamento. Este exemplo pode ser implementado utilizando a proposta de interface de conexão de *cores*, sendo cada uma das quatro configurações um *core* comunicando-se com a *interface*.

Julga-se essencial o desenvolvimento desta interface de comunicação de *cores* para que seja possível a criação de ferramentas de projeto para reconfiguração dinâmica de sistemas digitais.

## 6. Conclusões e trabalhos futuros

Este trabalho revisou os conceitos relacionados a *cores*, enfatizando sua importância no mercado atual de semicondutores, o qual exige projetos cada vez mais complexos e com o *time-to-market* cada vez mais curto. Somente através da reutilização de módulos pré-projetados, pré-testados e otimizados (*cores*) é possível atender a estas exigências.

Três trabalhos em andamento relacionados ao desenvolvimento de *cores* foram apresentados. O **primeiro** é relacionado a um conjunto de ferramentas que permite a projetista disponibilizar seus projetos pela Internet de uma forma organizada, facilitando, assim, a utilização destes projetos por parte de outros usuários. O **segundo** trabalho é relacionado a reconfiguração de *hard cores*, para sua distribuição. Como mencionado no início do artigo, *hard cores* são aqueles que apresentam a maior proteção à propriedade intelectual. O **terceiro** trabalho é a proposta de generalizar a reconfiguração de *hard cores*, através do desenvolvimento de uma interface de comunicação interna ao FPGA.

A meta deste trabalho é a convergência dos três trabalhos em andamento, ou seja, uma vez operante a estrutura de barramento, poder-se-á "encaixar" módulos de hardware neste barramento, configurando-os através das páginas de distribuição de *cores*. Desta forma teremos um conceito de hardware semelhante ao conceito de memória virtual, ou seja, em nosso hardware teremos apenas os módulos que precisam ser executados naquele momento. Os demais estão armazenados externamente, para uso posterior.

## 7. Bibliografia

- [1] SIA - Semiconductor Industry Association, 1999. Capturado em Jul. 2000. Online. Disponível na Internet [http://notes.sematech.org/1999\\_SIA\\_Roadmap/Home.htm](http://notes.sematech.org/1999_SIA_Roadmap/Home.htm)
- [2] SCHALLER, R. R. Moore's Law: Past, Present and Future. IEEE Spectrum, 34 (6): 52-59, June 1997.
- [3] CALAZANS, N. L. V. Projeto Lógico Automatizado de Sistemas Digitais Sequenciais. Imprinta Gráfica e Editora Ltda. 11a. Escola de Computação, Universidade Federal do Rio de Janeiro - UFRJ, Rio de Janeiro, 20-24 July, 1998.
- [4] MAZZOR, S., LANGSTRAAT, P. A Guide to VHDL. Kluwer Academic Publishers, Norwell, MA, 1992.
- [5] THOMAS, D., MOORBY, P. The Verilog Hardware Description Language. Kluwer Academic Publishers, (1991).
- [6] GUPTA, R. K., ZORIAN, Y. Introducing Core-Based System Design. IEEE Design & Test of Computers, p. 15-25, October-December 1997.
- [7] MADISETTI, V. K., SHEN L. Interface Design for Core-Based Systems. IEEE Design & Test of Computers, p. 42-51, October-December 1997.
- [8] KEATING, M. Reuse methodology manual for system-on-a-chip designs. Second Edition. Kluwer Academic Publishers, Norwell, MA, 1999.
- [9] HAASE, J., OBERTHÜR, T., OBERWESTBERG, M. Design Methodology for IP Providers. In Reuse Techniques for VLSI Design, p. 49-62. Kluwer Academic Publishers, Norwell, MA, 1999.
- [10] MESQUITA, D., MORAES, F., PALMA, J. C., MOLLER, L., CALAZANS, N. Reconfiguração Parcial e Remota de Cores FPGAs. VII IBERCHIP Workshop. Montevideo, Uruguai. Março, 2001. Disponível na Internet [http://www.inf.pucrs.br/~dmesquita/interest/itens/iberchip\\_reconf.pdf](http://www.inf.pucrs.br/~dmesquita/interest/itens/iberchip_reconf.pdf)
- [11] XILINX INC. The JBits 2.4 SDK for Virtex. Capturado em Nov. 2000. Online. Disponível na Internet [ftp://customer:xilinx@ftp.xilinx.com/download/JBits2\\_4.exe](ftp://customer:xilinx@ftp.xilinx.com/download/JBits2_4.exe).
- [12] ARM Ltda. AMBA Specification Overview. Capturado em Jul. 2000. Online. Disponível na Internet <http://www.arm.com/Pro+Peripherals/AMBA>.
- [13] IBM. The CoreConnect™ Bus Architecture. Capturado em Jun. 1999. Online. Disponível na Internet [http://www.chips.ibm.com/products/coreconnect/docs/crcon\\_wp.pdf](http://www.chips.ibm.com/products/coreconnect/docs/crcon_wp.pdf)
- [14] VILLASENOR, J., W. H. Mangione-Smith. Configurable Computing. Scientific American, pp. 54-59, June 1997.