Revista de Sistemas de Informação da FSMA
n. 20 (2017) pp. 38-45

SISTEMAS DE INFORMAÇÃO
http://www.fsma.edu.br/si/sistemas.html

TRILHA PRINCIPAL

NovaCORE vFPGA: virtualização e reconfiguração instantânea

Andre Bannwart Perina, *Doutorando em Ciências de Computação e Matemática Comp., ICMC-USP*,
Jesimar da Silva Arantes, *Doutorando em Ciências de Computação e Matemática Comp., ICMC-USP*,
Vanderlei Bonato, *Professor Associado, ICMC-USP*

***Resumo*—Apesar de sua notória importância no ramo da computação paralela, *Field-Programmable Gate Arrays* (FPGAs) tem um espaço de aplicação muito mais restrito se comparado às tecnologias de programação facilitada e custo reduzido como *Graphic Processing Units* (GPUs) ou processadores *Manycores*. No entanto, a tecnologia dos FPGAs tem funcionalidades diferenciadas que podem trazer benefícios únicos para determinadas aplicações, como por exemplo a reconfiguração parcial dinâmica. Este benefício, porém, pode apresentar tempos de reconfiguração proibitivos (milissegundos) para certas aplicações devido a sua demora. Este artigo apresenta uma nova arquitetura de um FPGA virtual simples capaz de armazenar diferentes configurações (contextos) e com capacidade de reconfiguração instantânea de seu *hardware* (em um único ciclo de clock). Experimentos mostraram que o armazenamento de diferentes contextos é uma solução viável, pois 4 contextos armazenados simultaneamente aumentaram em média apenas 2× a quantidade de recursos lógicos. Por outro lado, a sobrecarga da virtualização - a relação entre o número de elementos lógicos físicos utilizados e o número de elementos virtuais implementados - foi em média de 269,9× para experimentos com apenas 1 contexto.**

***Palavras-chave*—FPGA, virtualização, reconfiguração dinâmica, reconfiguração instantânea, computação reconfigurável.**

NovaCORE vFPGA: virtualisation and immediate reconfiguration

***Abstract*—Although quite important in the parallel computing area, Field-Programmable Gate Arrays (FPGAs) have a restricted application space when compared to other technologies such as Graphic Processing Units (GPUs) or manycore processors. However, FPGAs have some specific features (e.g. dynamic partial reconfiguration) that may benefit certain applications. Dynamic partial reconfiguration, however, may present prohibitive times (milliseconds). This article presents a simple virtual FPGA architecture capable of holding different configurations (contexts) and also capable of switching between those contexts in a single clock cycle. Experiments have shown that storing 4 different contexts increased in 2× the amount of used logic resources. However, the virtualisation overload - relation between used physical logic resources and the implemented virtual logic resources - was 269.9× in average for experiments with just 1 context.**

Autor correspondente: Andre Bannwart Perina, *abperina@usp.br*

***Index Terms*—FPGA, virtualisation, dynamic reconfiguration, immediate reconfiguration, reconfigurable computing.**

I. Introdução

NAS últimas décadas, várias arquiteturas de computação foram investigadas para resolver uma grande gama de problemas. Primeiramente, computadores digitais se consolidaram à frente dos computadores analógicos. Com o passar dos anos, as tendências apontaram para um *hardware* fixo, com o objetivo de facilitar a programação, reduzir custo de produção e aumentar a reusabilidade com possíveis penalidades em consumo energético e performance. Atualmente, há várias arquiteturas de *hardware* fixo consolidadas, por exemplo processadores de propósito geral (*Central Processing Units*, ou CPUs) e processadores vetoriais.

Processadores de propósito geral foram demonstrados em 1945 pelo matemático John von Neumann como um *hardware* fixo e simples, capaz de executar uma sequência de instruções e processar dados sequencialmente [1] [2]. Como forma de permitir uma computação mais paralela, as CPUs evoluíram de diversas maneiras, desde a paralelização ao nível de *pipeline*, até a inclusão de núcleos de processamento paralelos (como é o caso dos *Manycores*) para processar diferentes segmentos de dados.

Processadores vetoriais são uma extensão a CPUs tal que o processamento é realizado em vetores de dados paralelamente (modelo conhecido como *Single Instruction Multiple Data*, ou SIMD). O mais famoso representante desta família são as GPUs (*Graphics Processing Units*). Criada em meados de 1980 para processamento gráfico, a arquitetura evoluiu para permitir o processamento de qualquer tipo de dado vetorial, junto com a criação de linguagens e plataformas para facilitarem a programação destas arquiteturas, como NVIDIA CUDA [3] ou Khronos OpenCL [4]. GPUs são conhecidas por consumirem uma quantidade alta de energia, ter custo moderado e facilidade de programação paralela.

Além dos processadores de *software* que possuem um *hardware* fixo, existe também a tecnologia de *hardware* reconfigurável, como por exemplo FPGAs. Composto por um vetor de elementos lógicos e uma malha de interconexão configuráveis, são capazes de mapear circuitos

lógicos e provêm paralelismo a nível de *hardware*, têm baixo consumo energético (quando comparado a CPUs e GPUs) e permitem a reconfiguração do *hardware* em campo. Porém, FPGAs têm um custo de moderado a alto, não são triviais para serem programadas e o tempo de compilação (síntese) pode chegar a horas ou dias.

As arquiteturas de FPGAs modernas podem alterar o circuito lógico programado em tempo de execução, como é o caso da reconfiguração parcial dinâmica. Este recurso permite que parte de um FPGA possa ser reconfigurada com outro subcircuito sem congelar o todo, permitindo maior flexibilidade. Esta tecnologia é utilizada, por exemplo, com a ferramenta Intel FPGA SDK for OpenCL [5], na qual uma parte do FPGA é mantida constante (módulos de comunicação e gerenciamento), enquanto uma pequena região é constantemente reprogramada (aplicação).

Este artigo apresenta um sistema de virtualização em FPGA. De acordo com [6], a virtualização ocorre quando uma aplicação é executada em *hardware* virtualizado em oposição ao físico. Isto é, ao invés da aplicação ser diretamente implementada em FPGA, uma camada adicional é implementada (camada virtual), esta por fim recebendo a aplicação. A principal vantagem da virtualização é permitir uma customização ainda maior que a possível em FPGAs físicos. Por outro lado, tal camada adiciona uma sobrecarga de recursos lógicos (o sistema aplicação + virtualização faz uso de mais recursos lógicos do que implementar a aplicação diretamente no FPGA físico).

No caso da reconfiguração dinâmica disponível em FPGAs físicos, o tempo de reconfiguração é na ordem de milissegundos, o que invibializa a sua aplicação em diversos problemas com restrições de tempo real. Com a arquitetura aqui apresentada, é possível armazenar diferentes configurações de circuitos lógicos em componentes virtuais, implementados a partir de recursos físicos de um FPGA real. Deste modo, é possível intercalar entre estas diferentes configurações (aqui denominadas dimensões) em apenas um ciclo de clock, o que corresponde a um tempo na ordem de nanossegundos.

O artigo está estruturado como segue: a Seção II apresenta o referencial teórico relacionado à virtualização de FPGAs e sistemas com armazenamento de várias configurações. A Seção III apresenta os problemas da reconfiguração em FPGAs. Em seguida, a Seção IV mostra a metodologia e a implementação empregada na virtualização. Resultados são expostos na Seção VI. Por fim, a Seção VII apresenta a conclusão e indica possíveis trabalhos futuros.

II. Referencial Teórico

A. Virtualização

A virtualização em FPGAs permite uma maior flexibilidade da arquitetura reconfigurável, sendo útil em casos onde o desenvolvedor tem a intenção de aplicar modificações impossíveis de serem realizadas em arquiteturas fechadas e proprietárias de FPGAs.

No trabalho de [7] foram estudados e discutidos conceitos relacionados à virtualização de FPGAs. Neste, a virtualização é vista como uma extensão do físico. Os autores afirmam que diferentes aplicações podem ser beneficiadas com o uso destas plataformas, como sistemas multimídia, algoritmos de voz e compressão de imagens.

Os autores em [6] identificaram três abordagens principais ao se efetuar a virtualização: particionamento temporal, execução virtualizada e máquina virtual. O particionamento temporal foi o primeiro estilo de virtualização investigado. A grande vantagem por trás desta técnica é a possibilidade de colocar uma aplicação de tamanho maior que o *hardware* reconfigurável disponibiliza. Para tanto, este método divide a aplicação em partes menores que se alocam sobre o dispositivo, sendo executadas sequencialmente. O trabalho de [8] é um exemplo prático deste conceito.

Ambos [9] e [10] destacam como grande vantagem o fato de que mudanças em *hardware* não alteram o nível virtual. Uma outra vantagem obtida é a exploração da reconfiguração parcial em plataformas que não suportam tal tecnologia.

Em [11], os autores apresentam um FPGA virtual denominado ZUMA. Através de diversas otimizações nos sistemas de roteamento e nos elementos lógicos, os autores conseguiram criar uma virtualização em que ocorre uma sobrecarga de 40×[1]. Este número, apesar de grande, é pequeno comparado com outras tentativas de virtualização. Os autores também desenvolveram ferramentas para assistir na programação desta arquitetura.

Em [12], o ZUMA foi expandido adicionando-se uma camada intermediária para permitir uma comunicação entre circuitos implementados no FPGA físico e no FPGA virtual, além da adicão de suporte para circuitos sequenciais. Com estas expansões, a sobrecarga aumentou para 66×.

B. FPGAs com Troca de Contexto

OS FPGAs com troca de contexto são arquiteturas nas quais diferentes configurações de circuitos residem no chip, podendo ser trocadas instantaneamente.

O *Dynamically Programmable Gate Array*, ou DPGA, é um protótipo desta tecnologia [13]. Neste trabalho, os autores desenvolveram o modelo eletrônico de um DPGA, que pode ser fabricado com uso de tecnologia CMOS. Em [14], o mesmo conceito é apresentado como uma extensão para um FPGA da Xilinx. Um protótipo desta tecnologia também foi apresentado sob o nome *Context-Switching Reconfigurable Computing*, ou CSRC [15]. Aplicações para estas arquiteturas são apresentadas em [16] e [17].

III. Problema

UM FPGA é composto principalmente de três macrocomponentes: módulos lógicos (onde a lógica é armazenada), módulos de interconexão (utilizados para conectar a lógica) e os módulos de entrada e saída. No caso de um FPGA baseado em memória SRAM, os módulos

[1] A sobrecarga indica quantos elementos lógicos físicos são necessários para implementar um elemento lógico virtual.

são configurados através de bits salvos em memória, conservando a configuração enquanto o sistema estiver energizado. Esta configuração é implantada com o uso de um arquivo denominado *bitstream*. O envio deste arquivo para dentro do FPGA é lento, levando segundos em dispositivos complexos como o Intel FPGA Stratix V. No caso da reconfiguração parcial dinâmica este tempo é reduzido, pois é possível que apenas parte do FPGA seja reconfigurado, reduzindo o tamanho do *bitstream* e consequentemente o seu tempo de envio. É possível também que o *bitstream* já esteja presente dentro do FPGA em memórias internas, bastando transferi-lo para as memórias de configuração para que a reconfiguração seja efetuada. No entanto, a reconfiguração parcial ainda ocorre na ordem dos milissegundos, uma sobrecarga que para muitos casos inviabiliza a sua aplicação. No trabalho aqui apresentado, faz-se uso da virtualização para obter uma arquitetura de FPGA onde seja possível realizar reconfigurações instantâneas, reduzindo o tempo de milissegundos para nanossegundos.

IV. Metodologia e Implementação

UMA arquitetura tradicional de FPGA é composta de quatro componentes: *Configurable Logic Blocks* (CLBs), *Connection Blocks* (CBs), *Switch Blocks* (SBs) e blocos de entrada e saída (*I/O Blocks*).

Os CLBs são responsáveis por armazenarem funções lógicas (por exemplo funções do tipo E, OU, ou combinação delas. Isto é feito com o uso de *LookUp Tables* (LUTs)[2]. A Figura 1 apresenta o mapeamento de uma função lógica de 4 entradas em uma LUT. As entradas da função lógica são utilizadas como endereço da memória e a saída dela é o resultado da função lógica. Um CLB pode conter, além de uma LUT, um registrador que pode ser usado para armazenar dados intermediários.

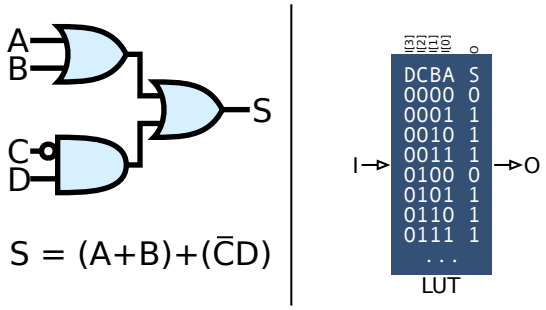

Fig. 1. Exemplo de uso de uma LUT. À esquerda, um simples circuito lógico e sua expressão. À direita, o mapeamento da tabela verdade deste circuito em uma LUT. As entradas A, B, C e D são utilizadas como endereço desta memória (entrada I na figura), enquanto o dado armazenado é a saída da expressão lógica (S). Apenas as 8 primeiras linhas são mostradas.

As entradas e saídas de um CLB são conectadas à uma malha de interconexão através de um *Connection Block*. Isto permite que diversos CLBs (cada um com diferentes circuitos lógicos) se comuniquem. Também é possível conectar os CLBs aos *I/O Blocks*, permitindo a interação do FPGA com agentes externos, como botões e LEDs. A malha de interconexão é composta de diversos canais horizontais e verticais. A conexão entre os canais ocorre em circuitos comutadores, denominados *Switch Blocks*. A maior parte dos FPGAs presentes hoje no mercado se baseiam neste conceitos, porém apresentam mais tipos de módulos fisicamente implementados (por exemplo: blocos de memórias para armazenamento de dados, somadores e multiplicadores presentes dentro dos CLBs, etc.) para permitir a implementação de circuitos mais elaborados.

A arquitetura desenvolvida (denominada NovaCORE) faz uso da virtualização em FPGA e considera o conceito de multidimensionalidade (várias configurações presentes *in loco*), com a finalidade de diminuir o tempo de reconfiguração parcial.

O conceito de multidimensionalidade é adquirido através do aumento do tamanho dos registradores de configuração do FPGA. Por exemplo, supondo que um multiplexador interno de um SB necessite de 1 bit para sua configuração, ao adicionar 3 bits, é possível armazenar 4 diferentes configurações. O mesmo ocorre com as LUTs internas dos CLBs, como mostra a Figura 2. Ao invés de armazenar apenas uma expressão lógica, várias são armazenadas e a escolha é feita através de um multiplexador externo à LUT. Ao adicionar a multidimensionalidade, as diferentes configurações já estarão localizadas em seus respectivos componentes, permitindo o chaveamento entre elas instantaneamente.

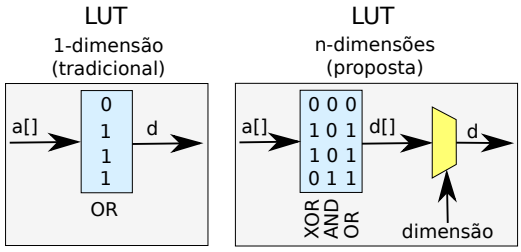

Fig. 2. Esquema de LUTs com única dimensão e múltiplas dimensões. No primeiro caso, apenas uma função é armazenada na LUT (função OU). No segundo caso, multiplas funções são armazenadas (OU exclusivo, E e OU), a escolha entre qual função utilizar é feita por um multiplexador conectado à saída da LUT.

A arquitetura completa é exposta na Figura 3. A disposição dos componentes foi baseada na arquitetura básica exposta por [18]. Desenvolvido em Verilog, o NovaCORE é altamente parametrizável tanto em quantidade de CLBs quanto em número de dimensões. Ao considerar apenas uma dimensão, toda a lógica de multidimensionalidade é removida, portanto é possível analisar a sobrecarga gerada pela lógica adicional para esta funcionalidade.

Para a lógica síncrona, o NovaCORE conta com um *clock* global fornecido externamente, não podendo ser gerado por lógica interna.

[2]LUTs são simples memórias assíncronas, ou seja, memórias que não utilizam *clock* para escrita e leitura.

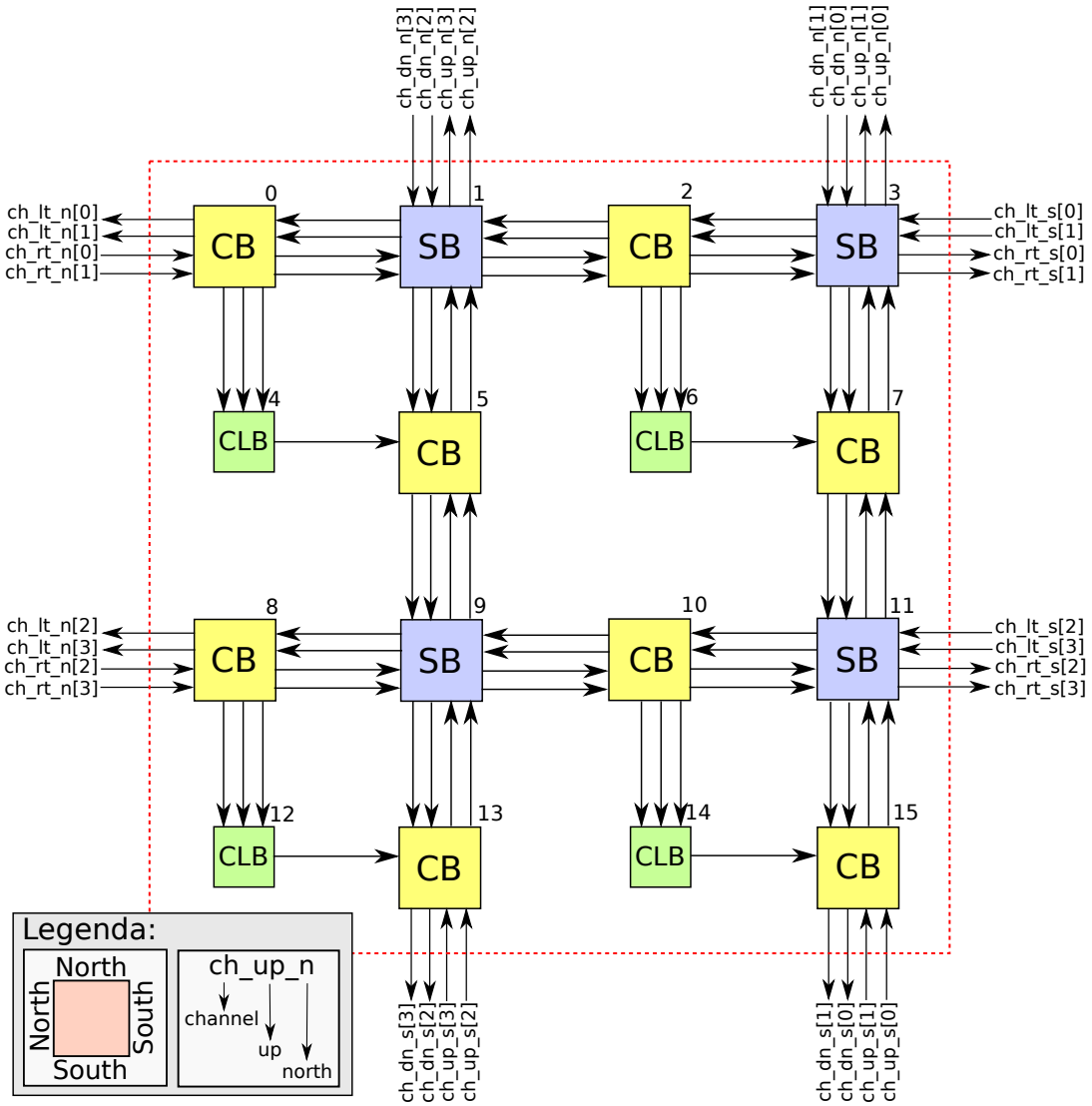

Fig. 3. Arquitetura do NovaCORE. As setas externas ao tracejado definem os pinos de entrada e saída.

A. *Configurable Logic Block*

A arquitetura do CLB é exposta na Figura 4. Diferente dos FPGAs modernos, este CLB é simplificado, composto por uma LUT e um registrador. Nesta implementação, o registrador é compartilhado entre as dimensões. A escolha da dimensão é feita através de um multiplexador externo à LUT.

Como memórias são elementos constantemente utilizados em circuitos lógicos e a sua implementação em FPGA pode usar uma grande quantidade de recursos, muitos FPGAs contam com blocos de memória já implementados e mapeáveis (*Block RAMs*, ou BRAMs). As memórias são síncronas, portanto inadequadas para a implementação de LUTs. Contudo, alguns FPGAs contam com memórias capazes de prover leituras de modo assíncrono. Os módulos do tipo MLAB dos FPGAs Altera Stratix III e IV permitem tal uso [19]. Com isto, é possível implementar os CLBs utilizando BRAMs, consequentemente reduzindo a utilização de recursos lógicos.

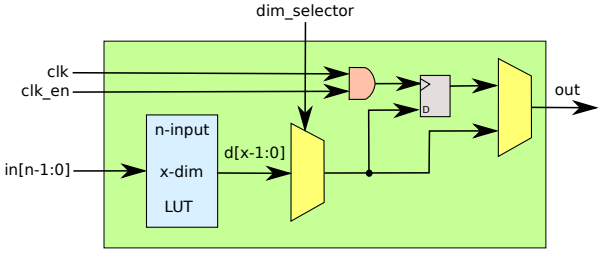

Fig. 4. Esquema de um CLB contendo uma LUT e um registrador. O multiplexador mais à esquerda é utilizado para escolher a dimensão da LUT, enquanto o multiplexador da direita é utilizado para selecionar se a saída do CLB será registrada (saída conectada ao registrador) ou não (saída conectada diretamente à saída da LUT).

B. *Connection Block*

O *Connection Block* é responsável por conectar as entradas e saídas dos CLBs aos canais de interconexão. A arquitetura básica é exposta na Figura 5. Há um registrador de configuração para cada conexão ao CLB

(blocos à direita na figura com o texto "bloco X: Y") e a configuração ocorre diferentemente para entradas e saídas:

- **CB → CLB:** O registrador define qual canal (da direita para a esquerda) será conectado;
- **CB ← CLB:** O registrador define quais canais receberão o dado, sendo o bit menos significativo para o canal mais à direita.

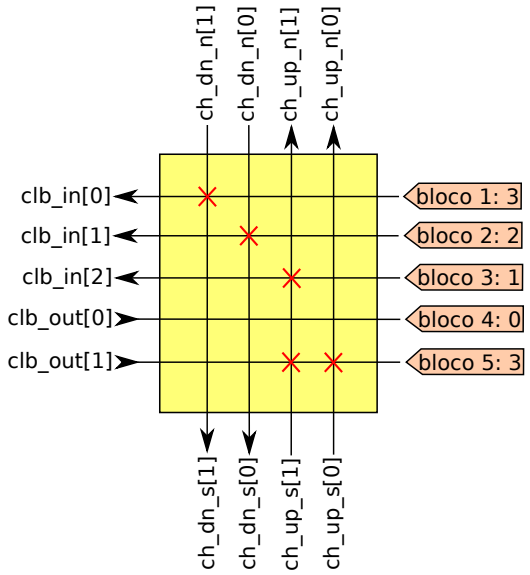

Fig. 5. Esquema detalhando o funcionamento do *Connection Block*. As setas horizontais indicam conexões ao CLB, enquanto as setas verticais indicam as conexões aos canais de interconexão. Um X na malha indica que tal ponto está conectado. À direita, são expostos 5 registradores de configuração e seus valores atuais, um para cada linha da malha.

V. Switch Block

A Figura 6 mostra o esquema de um SB. Para cada canal de saída, há um multiplexador recebendo todos os canais de entrada, permitindo que todas as entradas possam ser conectadas em todas as saídas. A numeração dos canais (separada para os canais de entrada e saída) ocorre da direita para a esquerda e de cima para baixo.

A. Programação

PARA programar o NovaCORE foi desenvolvido um sistema contendo um processador *softcore* NIOS II responsável por ler o *bitstream* (arquivo de configuração) de uma memória e introduzi-lo no FPGA. Tal sistema também é responsável por chavear as dimensões.

A programação é sequencial por pacotes. Cada um dos componentes (CLBs, CBs, SBs) conta com um identificador global[3] (*Unique ID*, UID). O dado é enviado em pacotes contendo o UID e as configurações específicas para cada tipo de componente. Pode ser necessário mais de um pacote para configurar completamente um componente

[3]Todos os blocos na Figura 3 contam com um identificador global, presentes nos cantos superiores direitos de cada bloco.

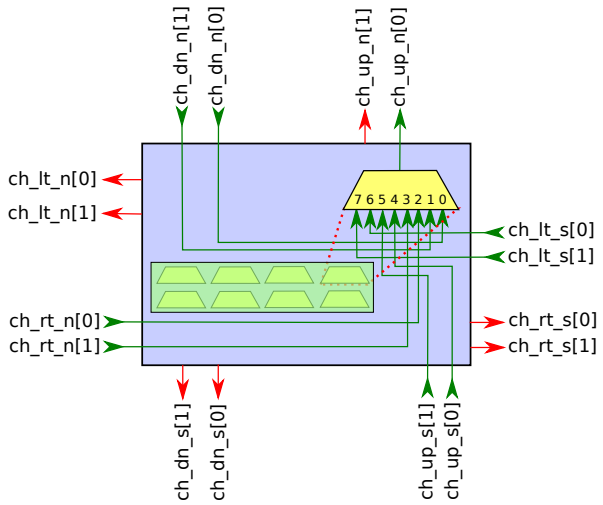

Fig. 6. Esquema interno de um *Switch Block*. Cada saída do bloco contém um multiplexador, onde este recebe todas as entradas do bloco. Isto permite que qualquer entrada seja ligada em qualquer saída. Nesta figura, apenas um multiplexador é completamente exposto.

(CBs, por exemplo, necessitam de um pacote para cada registrador de configuração).

Após a configuração, todas as dimensões estarão disponíveis dentro do FPGA. A reconfiguração, portanto, é instantânea, necessitando de apenas um ciclo de clock para ser efetivada. Na implementação atual, a reconfiguração é total e estática. Tal limitação não é impactante, pois:

- A reconfiguração parcial pode ser simulada através de duas dimensões que compartilhem configurações idênticas dos componentes fixos, permitindo inclusive partições mutantes de diversos formatos e tamanhos, característica inexistente em FPGAs atuais;
- A reconfiguração dinâmica pode ser simulada pelo fato da mudança de dimensão ocorrer em apenas um ciclo, não interrompendo de maneira impactante o funcionamento do sistema.

Até o momento, a geração do *bitstream* é manual: a configuração de cada registrador é descrita manualmente em um arquivo texto contendo os pacotes. O arquivo é traduzido com o uso de um *script* para um arquivo de inicialização de memória *on-chip*, sendo carregado no sistema de programação com o NIOS II.

VI. Resultados

PARA a validação dos resultados, os módulos foram implementados e testados separadamente com o uso do simulador Icarus Verilog [20]. Após a integração, o sistema (incluindo o subsistema de programação) foi sintetizado em um FPGA Intel Stratix IV (EP4SGX230KF40C2ES). A síntese foi realizada em um computador com processador Intel Xeon E5-1607, com 80GB de memória.

O sistema foi validado através da implementação de 4 simples funções lógicas (uma em cada dimensão) tendo botões, um clock como entradas e LEDs como saídas,

implementadas na configuração exposta na Tabela I. A largura do canal define quantas vias (subida e descida) há em cada canal de interconexão.

TABELA I
CONFIGURAÇÃO DE VALIDAÇÃO PARA O NOVACORE

CLBs	4
Largura Canais	4
Nro. Entradas LUT	2 + *clock_enable*
Dimensões	4

Para a análise de recursos lógicos (ALUTs[4] e Registradores) e frequência máxima para configuração e mudança de dimensão (f_{max}), diferentes configurações foram sintetizadas e seus resultados são expostos nas Tabelas II e III.

O tempo de reconfiguração total é calculado dividindo-se o número de ciclos necessários para a configuração pela frequência de configuração, fixado em 40MHz por ser um valor próximo ao limitante superior f_{max}. O número de ciclos (ou pacotes) necessários para a reconfiguração total é calculado pela soma de todos os pacotes necessários para a configuração de cada bloco da arquitetura, resultando na equação abaixo após simplificações:

$$n_{pkt} = n_{clb}((i_{clb}^2 + 1) + (2n_{ch}) + (i_{clb} + 2) + 2) \quad (1)$$

onde:

- n_{pkt}: número de pacotes;
- n_{clb}: número de CLBs;
- i_{clb}: número de entradas de um CLB;
- n_{ch}: tamanho dos canais.

TABELA II
RESULTADOS GERAIS DO NOVACORE (4 DIMENSÕES, LUT COM 2 ENTRADAS) NO FPGA FÍSICO

Nro. CLBs	2 × 2	5 × 5	7 × 7
Largura Canais	4	10	14
ALUTs	639	7017	28300
Registradores	654	6957	26098
f_{max} (MHz)	109,83	42,99	44,83
Reconf. total (40MHz)	$1,9\mu s$	$19,38\mu s$	$47,78\mu s$
Troca dim. (40MHz)	25ns	25ns	25ns

TABELA III
RESULTADOS GERAIS DO NOVACORE (1 DIMENSÃO, LUT COM 2 ENTRADAS) NO FPGA FÍSICO

Nro. CLBs	2 × 2	5 × 5	7 × 7
Largura Canais	4	10	14
ALUTs	320	4042	15025
Registradores	172	1465	5740
f_{max} (MHz)	132,86	44,18	65,42
Reconf. total (40MHz)	$1,9\mu s$	$19,38\mu s$	$47,78\mu s$

A fim de avaliar o aumento do uso de recursos lógicos em função da quantidade de CLBs da dimensão, apresenta-se na Tabela IV o resultado de síntese para os casos 9 × 9 e 10 × 10. Para estas dimensões maiores não foi possível realizar a análise temporal pelo fato da ferramenta de análise não suportar circuitos com este nível de complexidade. Por fim, a Figura 7 exibe um gráfico em relação ao uso de recursos lógicos para todas configurações exploradas neste trabalho. Como pode-se observar, ao aumentar em 4× o número de dimensões, o número de ALUTs físicas utilizadas aumentou em média $1,9\times$, enquanto o número de registradores aumentou em média $5,1\times$.

[4] *Adaptive LUTs*, um tipo avançado de LUTs.

TABELA IV
RESULTADOS DO NOVACORE PARA CONFIG. MAIORES (LUT COM 2 ENTRADAS) NO FPGA FÍSICO

	4 dimensões		1 dimensão	
Nro. CLBs	9 × 9	10 × 10	9 × 9	10 × 10
Largura Canais	18	20	18	20
ALUTs	32216	109690	15397	61112
Registradores	35662	69809	5337	12740

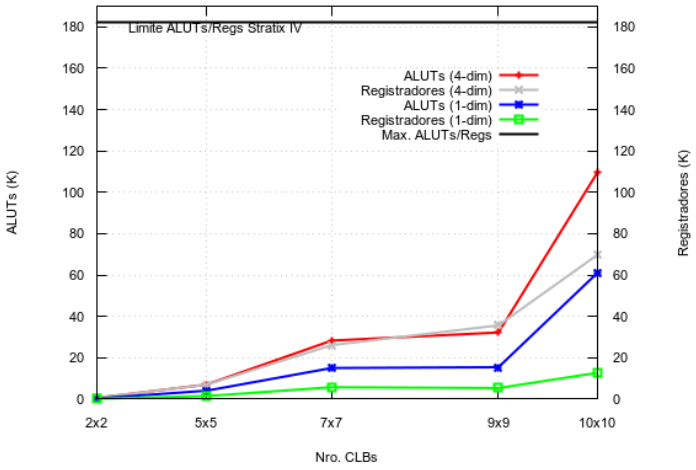

Fig. 7. Recursos lógicos utilizados pelas diferentes configurações. A linha horizontal superior mostra o limite em recursos (182400 ALUTs e registradores) do FPGA Stratix IV.

O aumento sublinear de elementos lógicos decorre da reutilização do barramento de interconexão entre as dimensões. Portanto, a dimensionalidade permite armazenar 4 contextos com um custo reduzido quando comparado a utilizar todos os contextos em paralelo em uma arquitetura tradicional. De acordo com os resultados, um simples FPGA virtual com 1 dimensão de 49 elementos lógicos virtuais ocupa 15025 elementos lógicos de um Stratix IV, resultando em uma sobrecarga de $\frac{15025}{49} = 306,63\times$. Considerando 4 dimensões totalizando 49×4 elementos lógicos, a sobrecarga é de $\frac{28300}{49\times4} = 144,39\times$. Porém, deve-se notar que ao utilizar 4 dimensões, há 49×4 elementos lógicos onde apenas 49 podem ser utilizados simultaneamente.

Comparando com [11], onde foi implementada uma arquitetura virtual sem multidimensionalidade e com uma sobrecarga de 40×, a arquitetura aqui apresentada de 1 dimensão ocupa 7,67 vezes mais recursos lógicos. A sobrecarga ocorre pela utilização de recursos lógicos para a implementação de componentes que fisicamente são fabricados apenas com transistores, como os CBs. Com

a multidimensionalidade proposta neste artigo foi possível demonstrar que o aumento de dimensões implica em um aumento sublinear de recursos lógicos, tendo o benefício de permitir a reconfiguração instantânea na ordem de nanossegundos, enquanto os FPGAs tradicionais (físicos ou virtuais) realizam reconfiguração parcial na ordem de milissegundos. Tal conceito, se fosse aplicado em FPGAs físicos, poderia trazer benefícios em aplicações onde reconfigurações instantâneas são desejáveis.

VII. Conclusão

MESMO com o advento da reconfiguração parcial dinâmica nos FPGAs (há alguns anos pela Xilinx e mais recente pela Altera), ainda há limitações que impedem seu uso de maneira generalizada, por complicações no particionamento físico e tempo excessivo de reconfiguração. Através da multidimensionalidade proposta no NovaCORE, é possível intercalar entre diferentes configurações de modo praticamente instantâneo, tendo em vista que o dado de configuração já se encontra *in loco*. A adição de tal funcionalidade aumenta a sobrecarga causada pela virtualização de maneira sublinear, o que mostra que tal artifício ainda é mais benéfico do que implementar todas as dimensões simultaneamente, sendo um ponto interessante de investigação pelas fabricantes.

Como trabalho futuro, espera-se integrar a arquitetura à ferramentas de síntese e *place and route*, como o MEANDER [21] e diminuir a sobrecarga excessiva através de técnicas como implementação das LUTs como uma combinação de multiplexadores. Também pretende-se aprimorar o barramento de reconfiguração de modo que este não escale de acordo com o número de CLBs.

VIII. Considerações Finais

OS autores desse artigo agradecem a Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) e a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro ao projeto de pesquisa. Agradecem também ao Laboratório de Computação Reconfigurável (LCR) do Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC-USP) pelo apoio técnico.

Todo o trabalho demonstrado neste artigo está disponível em um repositório GIT sob licença MIT, disponível em: https://github.com/comododragon/novacore.

Referências

[1] C. Bobda, *Introduction to Reconfigurable Computing: Architectures, algorithms, and applications*. Springer Science & Business Media, 2007.

[2] W. Stallings, *Computer Organization and Architecture: Designing for performance*. Pearson Education India, 2000.

[3] NVIDIA Corporation, *Plataforma de Computação Paralela*, Disponível em http://www.nvidia.com.br/object/cuda_home_new_br.html, acessado 11 dez. 2017, 2017.

[4] The Khronos Group Inc., *OpenCL Overview*, Disponível em https://www.khronos.org/opencl/, acessado 11 dez. 2017, 2016.

[5] Altera Inc., *Altera SDK for OpenCL - Overview*, Disponível em https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html, acessado 26 ago. 2016, 2016.

[6] C. Plessl e M. Platzner, "Virtualization of Hardware-Introduction and Survey", em *ERSA*, 2004, pp. 63–69.

[7] W. Fornaciari e V. Piuri, "Virtual FPGAs: some steps behind the physical barriers", em *Parallel and Distributed Processing*, Springer, 1998, pp. 7–12.

[8] M. Ullmann, M. Hübner, B. Grimm e J. Becker, "An FPGA run-time system for dynamical on-demand reconfiguration", em *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, IEEE, 2004, p. 135.

[9] P. Figuli, M. Hübner, R. Girardey, F. Bapp, T. Bruckschlögl, F. Thoma, J. Henkel e J. Becker, "A heterogeneous SoC architecture with embedded virtual FPGA cores and runtime Core Fusion", em *Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on*, IEEE, 2011, pp. 96–103.

[10] M. Hübner, P. Figuli, R. Girardey, D. Soudris, K. Siozios e J. Becker, "A heterogeneous multicore system on chip with run-time reconfigurable virtual FPGA architecture", em *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, IEEE, 2011, pp. 143–149.

[11] A. Brant e G. G. Lemieux, "ZUMA: An open FPGA overlay architecture", em *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*, IEEE, 2012, pp. 93–96.

[12] T. Wiersema, A. Bockhorn e M. Platzner, "An architecture and design tool flow for embedding a virtual FPGA into a reconfigurable system-on-chip", *Computers & Electrical Engineering*, vol. 55, pp. 112–122, 2016.

[13] E. Tau, D. Chen, I. Eslick e J. Brown, "A first generation DPGA implementation", em *Proceedings of the Third Canadian Workshop on Field-Programmable Devices*, Citeseer, 1995.

[14] S. Trimberger, D. Carberry, A. Johnson e J. Wong, "A time-multiplexed FPGA", em *Field-Programmable Custom Computing Machines, 1997. Proceedings., the 5th Annual IEEE Symposium on*, IEEE, 1997, pp. 22–28.

[15] S. M. Scalera e J. R. Vázquez, "The design and implementation of a context switching FPGA", em *FPGAs for Custom Computing Machines, 1998. Proceedings. IEEE Symposium on*, IEEE, 1998, pp. 78–85.

[16] A. DeHon, "DPGA utilization and application", em *Proceedings of the 1996 ACM fourth internatio-*

nal symposium on Field-programmable gate arrays, ACM, 1996, pp. 115–121.

[17] K. Puttegowda, D. I. Lehn, J. H. Park, P. Athanas e M. Jones, "Context switching in a run-time reconfigurable system", *The Journal of Supercomputing*, vol. 26, n.º 3, pp. 239–257, 2003.

[18] S. Hauck e A. DeHon, *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007, ISBN: 9780080556017, 9780123705228.

[19] Altera Inc., *Can Quartus II synthesis infer a small asynchronous memory structure that can be placed in a Stratix III MLAB block?*, Disponível em https://www.altera.com/support/support-resources/knowledge-base/solutions/rd12112006_212.html, acessado 23 nov. 2017, 2007.

[20] IcarusVerilog, *Icarus Verilog*, Disponível em http://iverilog.icarus.com/, acessado 13 mai. 2016, 2016.

[21] Meander, *Meander - FPGA Design Framework*, Disponível em http://proteas.microlab.ntua.gr/meander/, acessado 13 mai. 2016, 2016.

Andre Bannwart Perina (autor correspondente) é bacharel em Ciências de Computação pela Universidade de São Paulo, campus de São Carlos - SP. Tem experiência na área de Arquitetura de Sistemas de Computação e Computação Reconfigurável. Atualmente doutorando no programa de Ciências de Computação e Matemática Computacional pela Universidade de São Paulo, campus de São Carlos, sob financiamento da FAPESP. E-mail: *abperina@usp.br*.

Jesimar da Silva Arantes é doutorando em Ciência da Computação e Matemática Computacional (CCMC) na Universidade de São Paulo (USP). É mestre em ciências pela USP em 2016. É bacharel em Ciência da Computação pela Universidade Federal de Lavras (UFLA) em 2014. Possui graduação também em Sistemas de Informação pela UFLA em 2013. Tem interesse nas áreas de Algoritmos Evolutivos, Inteligência Computacional, Robótica, Veículos Aéreos Não-Tripulados e Desenvolvimento de Jogos.

Vanderlei Bonato é Bacharel (2002), Mestre (2004) e Doutor (2008) em Ciência de Computação. Ele obteve o título de Doutor pelo Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC-USP), o qual foi parcialmente desenvolvido no Departamento de Engenharia Elétrica e Eletrônica do Imperial College London. Sua área de interesse é arquitetura de computação reconfigurável, incluindo ferramentas para modelagem e síntese. O mesmo participa do comitê de programa de diversos eventos nacionais e internacionais, como o FPL, ARC, WRC, ERAD-SP, FEEC e SBESC, e atua como revisor ad hoc para revistas científicas e agências de fomento. Trabalha como professor no ICMC-USP desde 2009 onde obteve o seu título de Livre-Docência em 2014.