UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E INFORMÁTICA INDUSTRIAL

DANIEL MEALHA CABRITA

SIMULADOR DE ALTA VELOCIDADE EM FPGA DE CIRCUITOS LUT DE LÓGICA COMBINACIONAL DE TOPOLOGIA ARBITRÁRIA PARA ALGORITMOS EVOLUCIONÁRIOS

DISSERTAÇÃO

CURITIBA

2015

DANIEL MEALHA CABRITA

SIMULADOR DE ALTA VELOCIDADE EM FPGA DE CIRCUITOS LUT DE LÓGICA COMBINACIONAL DE TOPOLOGIA ARBITRÁRIA PARA ALGORITMOS EVOLUCIONÁRIOS

Dissertação apresentada ao Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do grau de "Mestre em Ciências" – Área de Concentração: Engenharia de Automação e Sistemas.

Orientador: Prof. Dr. Carlos Raimundo Erig Lima

Co-orientador: Prof. Dr. Walter Godoy Jr. (in memoriam)

CURITIBA

2015

Dados Internacionais de Catalogação na Publicação

C117s 2015

Cabrita, Daniel Mealha
Simulador de alta velocidade em FPGA de circuitos LUT de lógica combinacional de topologia arbitrária para algoritmos evolucionários / Daniel Mealha Cabrita.-- 2015.
70 f.: il.; 30 cm

Texto em português, com resumo em inglês.
Dissertação (Mestrado) - Universidade Tecnológica Federal do Paraná. Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial, Curitiba, 2015.
Bibliografia: f. 66-70.

1. Lógica combinatória. 2. Arranjos de lógica programável em campo. 3. Algoritmos genéticos. 4. Computação evolutiva. 5. Programação evolucionária (Computação). 6. Programação genética (Computação). 7. Sistemas de computação virtual. 8. Sistemas de computação adaptativos. 9. Eletrônica digital. 10. Simulação (Computadores digitais). 11. Engenharia elétrica - Dissertações. I. Lima, Carlos Raimundo Erig, orient. II. Godoy Júnior, Walter. III. Universidade Tecnológica Federal do Paraná - Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial. IV. Título.

CDD 22 -- 621.3

Biblioteca Central da UTFPR, Câmpus Curitiba

Título da Dissertação Nº. ____

SIMULADOR DE ALTA VELOCIDADE EM FPGA DE CIRCUITOS LUT DE LOGICA COMBINATIONAL DE TOPOLOGIA ARBITRARIA PARA ALGORITMOS EVOLUCIONARIOS

por

Daniel Mealha Cabrita

Orientador: Prof. Dr. Carlos Raimundo Erig Lima

Esta dissertação foi apresentada como requisito parcial à obtenção do grau de MESTRE EM CIÊNCIAS – Área de Concentração: Eng. De Automação e Sistemas do Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial – CPGEI – da Universidade Tecnológica Federal do Paraná – UTFPR, às 14:00h do dia 26 de fevereiro de 2015. O trabalho foi aprovado pela Banca Examinadora, composta pelos professores doutores:

Prof. Dr. Carlos Raimundo Erig Lima
(Presidente – UTFPR)

Prof. Dr. Oscar da Costa Gouveia Filho
(UFPR)

Prof. Dr. Jean Marcelo Simão
(UTFPR)

Visto da coordenação:

Prof. Emilio Carlos Gomes Wille, Dr.
(Coordenador do CPGEI)

A Folha de Aprovação assinada encontra-se na Coordenação do Programa de Pós Graduação de Engenharia Elétrica e Infomática Industrial.

AGRADECIMENTOS

Primeiramente, registro meu grande agradecimento ao Prof. Walter Godoy Jr., meu orientador original e mentor, o qual acreditou no meu potencial e não poupou esforços quando estive em dificuldades e que, infelizmente, faleceu antes da conclusão do meu mestrado.

Também agradeço ao Prof. Carlos Raimundo Erig Lima, meu orientador, que mostrou-se muito hábil em me direcionar para a criação de um trabalho de qualidade, mesmo com o pouco tempo disponível.

Agradeço ao Kamil, de Kraków, pela inspiração por ser um pesquisador por excelência e ser, ainda assim, completamente casual a respeito.

E, finalmente, agradeço ao meu amigo Marcos, companheiro de outros projetos e quem semeou a idéia de fazer o mestrado.

Літво, Ойчизно моя! ти єстесь як здровє;
Іле цѣа тжеба ценіць, тен тилко сѣа довє,
Кто цѣа страціў. Дзісь пѣакносьць твѫ в цаўей оздобє
Відза і опісуѣа, бо тѫскніа по тобє.
(Adam Mickiewicz)

RESUMO

MEALHA CABRITA, Daniel. SIMULADOR DE ALTA VELOCIDADE EM FPGA DE CIRCUITOS LUT DE LÓGICA COMBINACIONAL DE TOPOLOGIA ARBITRÁRIA PARA ALGORITMOS EVOLUCIONÁRIOS. 70 f. Dissertação – Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná. Curitiba, 2015.

Este trabalho apresenta uma arquitetura para simulação de circuitos de lógica combinacional de topologia arbitrária, visando interfaceamento com algoritmos evolutivos para fins de geração de hardware. A implementação é em FPGA utilizando a técnica VRC. O simulator permite circuitos compostos por LUTs de número de entradas parametrizável. A livre interconectividade entre as LUTs permite a construção de circuitos cíclicos. A arquitetura é modular e de interfaceamento simples. Alta performance é obtida através do uso de múltiplos módulos de simulação em paralelo, trazendo resultados que ultrapassam os obtidos em outros trabalhos utilizando DPR.

Palavras-chave: lógica combinacional, simulador de circuito, LUT, FPGA, algoritmos evolucionários, hardware evolutivo, programação genética, circuito virtual reconfigurável

ABSTRACT

MEALHA CABRITA, Daniel. A FAST SIMULATOR IN FPGA FOR LUT-BASED COMBINATIONAL LOGIC CIRCUITS OF ARBITRARY TOPOLOGY FOR EVOLUTIONARY ALGORITHMS. 70 f. Dissertação – Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial, Universidade Tecnológica Federal do Paraná. Curitiba, 2015.

This work presents an architecture for simulation of combinational logic circuits of arbitrary topology, meant to be interfaced with evolutionary algorithms for hardware generation. It was implemented in FPGA using the VRC technique. The simulator allows for circuits composed of LUTs of parametrizable number of imputs. The free interconectivity between LUTs allows the construction of cyclic circuits. The architecture is modular and of simple interfacing. High performance is obtained by the use of multiple simulation modules in parallel, bringing results that surpass the ones obtained from other works based on DPR.

Keywords: combinational logic, circuit simulator, LUT, FPGA, evolutionary algorithms, evolving hardware, genetic programming, virtual reconfigurable circuit

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

CBUS	barramento de configuração (configuration bus)
CC	circuito cíclico
CCLC	circuitos cíclico de lógica combinacional (cyclic combinational logic circuit)
CCLK	clock de configuração (configuration clock)
CCS	seção de configuração do circuito (circuit configurator section)
CFW	largura de preenchimento do circuito (circuit filling width)
CGP	programação genética cartesiana (genetic cartesian programming)
CIW	largura de entrada do circuito (circuit input width)
CLC	circuito de lógica combinacional (combinational logic circuit)
COW	largura de saída do circuito (circuit output width)
CPLD	dispositivo lógico programável complexo (complex programmable logic device)
CTL	camada de tradução de cromossomo (chromosome translation layer)
DC	duty cycle
DPR	reconfiguração dinâmica parcial (dynamic partial reconfiguration)
EA	algoritmo evolucionário (evolutionary algorithm)
EHW	hardware evolutivo (evolving hardware)
FE	avaliador de fitness (fitness evaluator)
FF	função de fitness (fitness function)
FPGA	arranjo de portas programável em campo (field-programmable gate array)
FS	seção de preenchimento (filling section)
GC	cromossomo genérico (generic chromosome)
GCS	simulador genérico de circuitos (generic circuit simulator)
GEP	programação por expressão de genes (gene expression programming)
GP	programação genética (genetic programming)
HDL	linguagem de descrição de hardware (hardware description language)
IBUS	barramento de entrada (input bus)
ICAP	porta de acesso para configuração interna (internal configuration access port)
IS	seção de entrada (input section)
JTAG	Joint Test Action Group
LE	elementos lógicos (logic elements)
LUT	look-up table
MBUS	barramento local ao módulo (local module bus)
MCCLK	clock de configuração de módulo (module configuration clock)
MC	controlador principal (main controller)
MSAD	decodificador de endereço de seleção de módulo (module selection address decoder)
MSA	endereço de seleção de módulo (module section address)
MUX	multiplexador
NoC	rede no chip (network on chip)
OS	seção de saída (output section)
PR	reconfiguração parcial (partial reconfiguration)
RD	dado para registro (register data)

SCLK	clock de simulação (simulation clock)
SECRT	tempo de evolução de um único ciclo (single evolution cycle run time)
SoC	system on a chip
TC	circuito de teste (test circuit)
TECC	tempo de gerações até convergência (total evolution cycles to convergence)
TEHW	target EHW module
VBS	vetor de estados de bit (vector of bit states)
VCS	seção do circuito virtual (virtual circuit section)
VHDL	linguagem de descrição de hardware do VHSIC (VHSIC Hardware Description Language)
VLUT	look-up table virtual (virtual look-up table)
VLUT MASK	máscara de bits da LUT (LUT bit mask)
VRC	circuito virtual reconfigurável (virtual reconfigurable circuit)
VRCM	módulo de circuito virtual reconfigurável (virtual reconfigurable circuit module)
VSG	vetor de portas simuladas (vector of simulated gates)

SUMÁRIO

1 INTRODUÇÃO

A otimização de circuitos de lógica combinacional começou com o método tradicional de mapas de Karnaugh (1953), que consiste em um processo manual e intuitivo de busca visual de padrões em dados colocados em tabelas, sendo limitado a obter circuitos booleanos compostos por operadores AND e OR.

A otimização citada acima consiste na redução de alguma característica do circuito de lógica combinacional, mantendo a tabela verdade inalterada, como a redução do número total de operadores booleanos, latência de resposta do circuito, consumo de área de silício ou mesmo, em caso de otimização multi-objetivo, de múltiplas características simultaneamente.

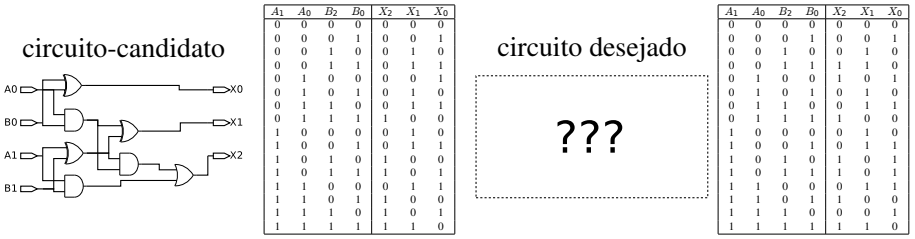
Posteriormente foram desenvolvidos novos métodos para otimização de circuitos de lógica combinacional. Porém, excetuando-se o uso de busca exaustiva que demanda um tempo excessivo de processamento, não foi desenvolvido um método determinístico para otimização multiobjetivo de circuitos de lógica combinacional, ou seja, uma forma de gerar circuitos otimizados considerando múltiplos parâmetros simultaneamente: número de portas lógicas, número de transistores, tempo de resposta, consumo de energia, área de silício etc.

Koza (1992) propôs o uso de programação genética com o uso de operadores booleanos distribuídos em uma árvore binária. O uso de algoritmos evolucionários para geração de circuitos de lógica combinacional despertou interesse por parte dos pesquisadores e, nos anos a seguir, surgiram diversos trabalhos visando criar geradores de circuitos optimizados como o de Coello et al. (2000) e, mais recentemetente, o de Wang et al. (2013).

O funcionamento dos algoritmos evolucionários, como algoritmos genéticos ou programação genética, consiste na geração de múltiplas soluções intermediárias ao longo de ciclos ou gerações, gradativamente obtendo soluções mais corretas. No momento em que uma em solução em particular atinja o critério de suficiência, o algoritmo é interrompido e o problema é considerado resolvido.

Os algoritmos evolucionários, de uma forma geral, geram soluções intermediárias que são sub-óptimas ou até mesmo inválidas. No caso de um circuito de lógica combinacional é

predominante a geração de specimens com uma tabela-verdade onde apenas parte das entradas sejam corretas, conforme exemplo mostrado na Fig. 1, que mostra um circuito-candidato e sua tabela-verdade, ao lado do circuito desejado do qual é conhecida *apenas* a sua tabela-verdade. Para calcular o fitness ou "pontuação genética" de um indivíduo, é necessário levantar o quanto as saídas do circuito gerado estão corretas, podendo ser através da distância Hamming entre as tabelas-verdade. Para tal procedimento, é necessário realizar a simulação do circuito gerado para todas as possíveis combinações de entradas. O processamento de simulação de circuitos é computacionalmente oneroso, já que o mesmo deve ser repetido para cada specimen gerado, de dezenas ou centenas por geração, por centenas ou milhares de gerações.



A_1	A_0	B_2	B_0	X_2	X_1	X_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

A_1	A_0	B_2	B_0	X_2	X_1	X_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	1	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	1
1	0	1	1	1	0	0
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	0	0	1
1	1	1	1	1	1	0

Figura 1: Algoritmo Evolucionário: exemplo de solução-candidata ao lado do circuito desejado desconhecido, com suas respectivas tabelas-verdade.

Slowik e Bialko (2008) já haviam reconhecido como um problema a questão de velocidade de simulação do circuito de lógica combinacional para cálculo de Fitness, caracterizando este como um problema de escalabilidade. Nos últimos anos foram propostas soluções para aumento de performance, em particular as baseadas em FPGA.

Este trabalho propõe um simulador de circuitos de lógica combinacional de topologia arbitrária, de alta performance, implementado sobre FPGA, para uso com algoritmos evolucionários.

1.1 MOTIVAÇÃO

A questão de velocidade de simulação do circuito de lógica combinacional já havia sido levantada por Slowik e Bialko (2008). Este autor (CABRITA et al., 2013) escreveu sobre a falta de benchmarks de tais simuladores em artigos envolvendo geração de circuitos de lógica combinacional. Posteriormente, Wang et al. (2013) manifestou preocupação com a pouca quantidade de pesquisas visando aumento de performance do simulador.

Embora poucas, surgiram pesquisas propondo simuladores de circuitos de lógica combinacional implementados em FPGA, para uso com algoritmos evolucionários, sendo encontrados os trabalhos de Sekanina e Friedl (2004), Ke et al. (2010), Ren et al. (2011), Bonilla

e Camargo (2011) e Wang et al. (2013).

Todos estes trabalhos apresentam um simulador de circuitos de arquitetura restrita a circuitos compostos por LUT *(look-up table)* de 2 ou 3 entradas apenas, com restrições de interconectividade entre estas LUTs. Na maioria dos casos o circuito é composto por uma matriz contendo um número fixo e não parametrizável de LUTs, além de limitações de número de entradas e saídas do circuito simulado. Isto torna os simuladores atrelados a uma forma específica de tratar o problema a nível de algoritmo evolucionário e não reutilizáveis em outras pesquisas.

Um outro problema é decorrente do fato destes trabalhos tratarem conjuntamente o problema do algoritmo evolucionário com o simulador, apresentando benchmarks que falham ao não incluir o tempo cronológico, ou o fazem de forma composta incluindo o tempo de simulação com o tempo gasto pelo algoritmo evolucionário. Este fato torna difícil, senão impossível, avaliar qual o benefício em termos de performance decorrente da implementação do simulador em FPGA, se comparado a, por exemplo, uma implementação realizada inteiramente em software. Esta é uma questão importante, pois é necessário justificar a inconveniência da implementação em FPGA se comparado com a facilidade de fazer a mesma em software.

E, por fim, em trabalho anterior deste autor (CABRITA et al., 2013) um simulador implementado em software foi capaz de atingir performance comparável a de uma implementação realizada em FPGA (BONILLA; CAMARGO, 2011). Isto instiga a pesquisa do quanto é realmente possível obter de performance de simulação em FPGA, se comparado a uma implementação em software.

1.2 OBJETIVOS

Este trabalho visa propor uma arquitetura para simulação de circuitos de lógica combinacional, de topologia arbitrária, de alta performance, implementada em FPGA, conveniente para uso com algoritmos evolucionários, e de forma a atender os seguintes quesitos:

1. Generalismo dentro do escopo de circuitos de lógica combinacional, suportando LUTs de número variável de entradas, permitindo trabalhar com LUTs com mais entradas que as da arquitetura nativa do FPGA. Suporte a um número variável de sinais de entrada e saída. LUTs com liberdade de interconectividade a ponto de suportar topologias arbitrárias, incluindo circuitos cíclicos.

2. Suporte, pelo simulador e após síntese em FPGA, a reconfiguração do circuito simulado

a qualquer momento e por quantas vezes desejado.

3. Alta performance de simulação sem prejuízo do generalismo. Considerando aceitável o generalismo através de parametrização pré-síntese.

4. Arquitetura modular e de simples interfaceamento, que possa ser tratada como uma "caixa preta", facilitando reutilização do simulador em pesquisas diversas.

5. Operação de múltiplos simuladores em paralelo.

6. Implementação de um simulador puro, sem provisões específicas para algoritmo evolucionário.

7. Compatibilidade com qualquer FPGA.

1.3 ESTRUTURA DA DISSERTAÇÃO

Este trabalho está organizado em sete capítulos. O capítulo 2 provê uma contextualização histórica, visando localizar o trabalho, que é altamente especializado, em um contexto mais amplo. No capítulo 3 é descrito o estado da arte, incluindo a evolução recente das técnicas atualmente disponíveis, assim como uma apresentação de pontos ainda carentes de pesquisa. O capítulo 4 descreve a validação em software da arquitetura proposta, seguida pela descrição da arquitetura em hardware, implementada em FPGA. O capítulo 5 descreve a metodologia dos testes realizados, contendo algumas considerações a respeito de pontos que afetam os resultados. Os resultados, assim como a análise dos mesmos, estão no capítulo 6 . O capítulo 7 contém a conclusão, com considerações sobre os resultados da pesquisa, assim como a situação deste trabalho dentro do estado da arte.

2 CONTEXTUALIZAÇÃO HISTÓRICA

Considerando o foco altamente especializado deste trabalho, faz-se necessário prover uma contextualização mais ampla, além do estado da arte, para uma adequada compreensão da contribuição deste trabalho.

A otimização de CLCs (circuito de lógica combinacional) começou a ser contemplada com o método tradicional de mapas de Karnaugh (1953), que consiste em um processo manual e intuitivo de busca visual de padrões em dados colocados em tabelas. O método de Quine (1955)-Mccluskey (1956) é uma evolução dos mapas de Karnaugh ao ser conveniente para a implementação como algoritmo, permitindo que o processo seja automatizado e escalável. Ambas as formas, no entanto, possuem como limitante a capacidade de trabalhar com lógica booleana restrita a operações AND e OR.

2.1 O USO DE ALGORITMOS EVOLUCIONÁRIOS

Uma forma de tratar a questão de otimização de CLC é transformando em um problema de EHW *(evolving hardware)*, que é a utilizada em diversas pesquisas, incluindo esta. EHW consiste na geração de circuitos através de EAs *(evolutionary algorithm)* e sintetizados em dispositivos lógicos configuráveis (HIGUCHI et al., 1996). EA é uma forma alternativa de tratar problemas sem forma escalável de resolução óptima conhecida, empregável em casos muito diversos incluindo projeto de antenas de veículos espaciais (LOHN et al., 2004).

O funcionamento dos algoritmos evolucionários como algoritmos genéticos ou programação genética, consiste na geração de múltiplas soluções intermediárias ao longo de ciclos ou gerações, gradativamente obtendo soluções mais corretas. No momento em que uma solução em particular atinja o critério de suficiência, o EA é interrompido e o problema considerado resolvido. Este processo é explicado de forma detalhada por Koza (1992).

Koza (1992) propôs o uso de GP *(genetic programming)* com o uso de operadores booleanos distribuídos em uma árvore binária, conforme exemplo na Fig. 2. Em anos seguintes,

surgiram outras propostas utilizando codificação de circuito em árvore como o IGE-CSA(GAN et al., 2009), este com raízes em GEP *(gene expression programming)* (FERREIRA, 2006).

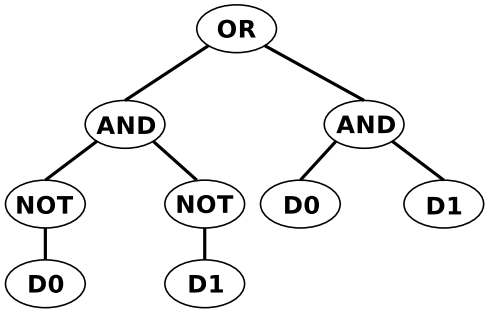


Figura 2: Exemplo de expressão booleana gerada por Programação Genética

Uma forma distinta foi proposta por Miller (1999) com refinamentos posteriores (MILLER et al., 2000), que é a CGP *(cartesian genetic programming)*, na qual os operadores são organizados como uma matriz de elementos livres e sem limitantes de interconectividade, conforme exemplo mostrado na Fig. 3. Variantes de CGP são comumente encontradas em trabalhos de síntese de CLC, com diversos níveis de restrições de interconectividade, incluindo estruturas orientadas em colunas (COELLO et al., 2003) e baseadas em vetor (SLOWIK; BIALKO, 2004).

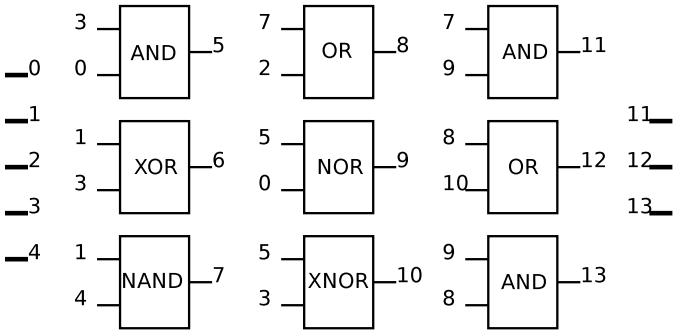


Figura 3: Exemplo de lógica combinacional gerada por Programação Genética Cartesiana

2.2 PROBLEMAS COM A VELOCIDADE CRONOLÓGICA DA EVOLUÇÃO

EHW tem limitações em termos de tempo cronológico, causados ambos pelo TECC *(total evolution cycles to convergence)* e pelo SECRT *(single evolution cycle run time)*. TECC é o número total de gerações passadas até o ponto em que é encontrada a solução e, em diversos trabalhos de EHW, é meramente referido como tempo de convergência.

SECRT é o tempo cronológico gasto em uma única geração. É particularmente crítico considerando o crescimento exponencial do tempo de simulação, gasto durante os testes de

cada CLC gerado, uma vez que é necessário levantar o comportamento do circuito para cada combinação possível em suas entradas. Por exemplo: para um CLC de 16 entradas, serão necessárias $2^{16} = 65536$ simulações do circuito para a avaliação de um único indivíduo. Considerando que uma população pode ter 100, 1000 ou mais indivíduos, e um número de gerações na ordem dos milhares, é visível a necessidade de efetuar simulações na ordem dos bilhões, ou mais, para tratar problemas de CLC. Assim, é perceptível a necessidade de um simulador de CLC de alta velocidade para a diminuição do tempo total gasto para a geração do circuito.

Slowik e Bialko (2008) reconheceram o problema da velocidade de simulação de CLC, particularmente a escalabilidade do simulador, tendo em vista o acréscimo de número de entradas do circuito refletir em um acréscimo exponencial do número de simulações necessárias para a verificação completa do CLC. O tempo de simulação dos CLCs afeta diretamente o SECRT. Para contornar este problema, foi proposto o uso de abordagens alternativas visando a redução do número de simulações necessárias durante a evolução do circuito.

O próximo capítulo aborda o estado da arte e descreve como diversos autores trataram o problema da velocidade de simulação de CLC.

3 ESTADO DA ARTE

3.1 DESCRIÇÃO GERAL

Em trabalho anterior (CABRITA et al., 2013) este autor escreveu sobre a falta de benchmarks de simuladores de CLC em artigos publicados sobre EHW, identificando o tempo gasto durante as simulações como um problema comumente negligenciado. Posteriormente, Wang et al. (2013) demonstrou preocupação com a pouca atenção dada ao problema de velocidade de simulação do circuito nas pesquisas publicadas. O problema da velocidade em si que ele expressou como SECRT, o qual inclui o tempo gasto pelas simulações de CLC. Os trabalhos atuais envolvendo geração de CLC, na maioria, publicam o TECC como valor de benchmark e raramente o SECRT.

Vasicek e Sekanina (2012) questionaram a escalabilidade ao realizar simulações completas de cada circuito candidato, com preocupação similar à manifestada por Slowik e Bialko (2008) anteriormente. Foi proposto avaliar apenas CLCs que apresentassem melhorias em algum critério, como o menor número total de portas lógicas, até sumariamente descartando soluções corretas no processo. A razão dada é que o terreno de fitness, o espaço em que a busca é realizada, é acidentado e desprovido de texturas diferenciadas. Embora tal lógica reduza o número de simulações por ciclo, isto meramente aceita o espaço de busca como intratável, assumindo a aparente aleatoriedade do terreno como um fato.

3.2 TÉCNICAS ANTERIORES PARA SÍNTESE DIRETA EM FPGA

Atualmente, a abordagem mais comum para reduzir o SECRT é aumentar a velocidade de simulação do circuito, em particular em implementações realizadas em FPGA *(field-programmable gate array)*. Reconfiguração nativa de FPGA para fins de simulação de CLC foi inicialmente proposto por Thompson (1996), o qual trabalhou com a atualmente obsoleta família Xilinx XC6200. A XC6200 era conveniente para uso com EA, pois o bitstream de reconfiguração permite ser arbitrariamente modificado, o dispositivo reconfigurado com este,

e com o hardware sintetizado pode ser realizada a verificação de comportamento do circuito-candidato utilizando as latências nativas do componente, ou seja, a resposta do circuito sob síntese direta sobre o FPGA sem o uso de abstrações estruturais. Embora este trabalho permitisse a construção de circuitos simples, os mesmos operavam em domínio analógico, sendo sensíveis a temperatura, tensão e variações do processo de fabricação do silício. Posteriormente a família XC6200 deixou de ser produzida, sendo substituída pelos Virtex. A família Virtex não dispõe de uma forma segura de reconfiguração utilizando bitstreams arbitrários, além de não possuir documentação suficiente da estrutura interna do componente, o que limita as possibilidades para uso com EHW.

Levi e Guccione (1999) propuseram o uso do JBits Software Development Kit da Xilinx para codificação dos circuitos para o formato proprietário de bitstream utilizado nos FPGA da linha Xilinx XC4000, para fins de EHW. O JBits é um software escrito em Java que permite a reconfiguração de baixo nível de FPGAs da Xilinx, sem a necessidade de conhecimento do formato do bitstream proprietário do componente (PATTERSON; GUCCIONE, 2001) .

O JBits tem como limitante a baixa velocidade de geração dos bitstreams, o que é problemático para EHW considerando o grande número de circuitos-candidatos que necessitam serem testados até ser obtida a solução. Oreifej et al. (2007) contornou tal problema eliminando o uso do JBits e gerando manualmente um circuito fixo, a ser usado como modelo, e restringindo as modificações a alterações no conteúdo das LUTs. Embora eliminando o overhead do JBits, a programação ainda se mostrou lenta devido às limitações de velocidade de programação serial através da porta JTAG *(Joint Test Action Group)*.

É oportuno informar que LUT, ou *look-up table*, é um componente de comportamento equivalente a um circuito de lógica combinacional, tipicamente com um número de entradas bastante reduzido e apenas um bit de saída. Uma LUT possui, internamente, uma memória contendo uma tabela-verdade programada pelo usuário, a qual define a saída binária da LUT para cada uma das possíveis combinações de entrada na mesma. LUTs são componentes internos básicos das FPGAs que, tipicamente, implementam LUTs de 4 ou 6 entradas.

3.3 DYNAMIC PARTIAL RECONFIGURATION

FPGAs da Xilinx suportam mecanismos de reconfiguração parcial PR *(partial reconfiguration)*, através de interfaces paralelas de alta velocidade. *Partial reconfiguration* consiste no recurso que permite a reconfiguração de partes do FPGA, sem modificação de

circuito das demais áreas. Tal recurso é particularmente útil para EHW, já que tipicamente os circuitos-candidatos a serem testados utilizam apenas uma fração da área do FPGA, o que permite uma transferência mais veloz dos mesmos. Um desses mecanismos é o SelectMap, da Xilinx (XILINX, 2009), que utiliza em uma interface paralela física de 8 a 32 bits, disponível através dos pinos externos do FPGA. Este recurso permite PR, que consiste na possibilidade de reconfigurar apenas partes do FPGA, enquanto mantém as demais regiões com o circuito inalterado. O SelectMap permite PR através de DPR *(dynamic partial reconfiguration)*, permitindo o funcionamento do restante do circuito durante a reconfiguração parcial.

Oreifej et al. (2007) considerou o uso do SelectMap em futuros trabalhos de EHW, mas o fato é que este recurso não despertou muito interesse para este fim. Uma das prováveis razões é a inconveniência da necessidade do uso de circuito externo ao FPGA, seja um microprocessador com um CPLD *(complex programmable logic device)*, ou um segundo FPGA. O SelectMap é academicamente obscuro de forma geral, sendo encontrado em trabalhos envolvendo correção de configuração de FPGA (HERRERA-ALZU; LOPEZ-VALLEJO, 2014), em particular para mitigar os efeitos de radiação ionizante (JACOBS et al., 2012), e como canal para ataque de segurança de FPGAs (HORI et al., 2012).

Os FPGAs topo-de-linha da Xilinx, a partir do Virtex-II Pro (CLAUS et al., 2007) passaram a oferecer suporte a DPR através do ICAP *(internal configuration access port)*. O ICAP, diferentemente do SelectMap, é uma interface interna no FPGA, mas similarmente permite DPR. Em um dos primeiros trabalhos utilizando ICAP, Kwok e Kwok (2004) propuseram o uso para implementar uma estrutura de circuitos modulares intercambiáveis conforme a demanda, no caso sendo usado para algoritmos de criptografia. Posteriormente o recurso ICAP passou a ser oferecido na linha de baixo custo da Xilinx, a partir do Spartan-6 (KOCH et al., 2010). Lopez et al. (2014) apresentou um sistema para EHW, baseado no ICAP do Spartan-6, com foco na capacidade de mensurar o consumo de energia do circuito-candidato, considerando como variável dentro do cálculo de fitness do EA.

Embora FPGAs com DPR estejam no mercado há anos, em diversas linhas da Lattice e Atmel (DONTHI; HAGGARD, 2003), tal recurso originalmente dependia de reprogramação através de componentes externos, tal como o SelectMap da Xilinx, sendo todos estes igualmente impopulares em pesquisas de EHW.

A Altera passou a disponibilizar DPR, de forma similar ao ICAP, a partir dos FPGAs topo-de-linha Stratix-V (ALTERA, 2010) (ICHINOMIYA et al., 2012), embora não tenha sido encontrado trabalhos que utilizem DPR da Altera para fins de EHW. Os FPGA Stratix-IV oferecem outro recurso genericamente chamado de Dynamic Reconfiguration, mas o mesmo

não se trata de DPR, mas específico à configuração dos transceivers (ALTERA, 2014).

No entanto, entre os trabalhos encontrados utilizando DPR para EHW, parece haver um monopólio de uso do ICAP da Xilinx, mesmo após alguns anos da introdução do Stratix-V.

Apesar do alto número de trabalhos utilizando DPR através de ICAP, ainda permaneceu o problema original, desde o final da produção do Xilinx XC6200, que é o fato do bitstream de reconfiguração do FPGA ser opaco, insuficientemente documentado, específico de cada fabricante e modelo de componente, além da possibilidade de circuitos inválidos causarem dano físico no componente. Conforme pode ser verificado nos trabalhos citados, se tornou padrão mitigar este problema através da geração manual de circuitos-modelo e limitar as modificações a alterações no conteúdo das LUTs, que podem ser identificadas no bitstream, mas sem efetuar mudanças de roteamento ou de outro tipo.

3.4 VIRTUAL RECONFIGURABLE CIRCUIT

Sekanina (2003) reconheceu as deficiências da reprogramação direta de FPGA, incluindo o relativamente longo tempo de reconfiguração, ainda que minimizado com o uso de DPR nativo do componente, e propôs uma estrutura de simulação de circuitos implementada sobre FPGA conhecida como VRC *(virtual reconfigurable circuit)*. VRC é uma abstração que consiste na implementação de um circuito reconfigurável sobre um FPGA. O VRC pode ser entendido, a grosso modo, como um FPGA implementado sobre FPGA.

Segundo Sekanina (2007), VRC é uma implementação de um circuito reconfigurável de domínio específico sobre um FPGA ordinário, permitindo a construção de um dispositivo reconfigurável perfeito, considerando as necessidades exatas do projetista. Nesta definição, ainda que aparentemente vaga, estão embutidos os pontos fortes do VRC, quando comparado à reconfiguração direta de FPGA, que é o caso do DPR. Os VRCs são programados exclusivamente com o mínimo de informação necessária para a descrição do circuito, dentro do espaço discreto de parâmetros permissíveis. Extendendo a analogia de FPGA-sobre-FPGA, a grosso modo pode ser dito que um VRC é um FPGA sob medida, implementado sobre um FPGA genérico.

Para fins de EHW, o bitstream para DPR deve ser alterado de forma a ser compatível com as especificidades arquiteturais do FPGA-destino. Na prática isso implica na necessidade de criação de um circuito-modelo, o qual é sub-óptimo pela necessidade de ser reconfigurável somente através das máscaras LUT. Mesmo assumindo que o espaço de variação de circuito esteja representado de forma compacta nas LUTs, deve ser considerado que durante a

reconfiguração via DPR, todo o circuito, e não somente as máscaras das LUTs, é transmitido. Considere-se que o espaço compacto de busca do CLC consiste apenas na definição da tabela das LUTs, assim como a definição das coordenadas de mapeamento das entradas destas mesmas LUTs. Ainda que o formato do bitstream para DPR fosse plenamente conhecido e manipulado de forma completa, o espaço não é compacto. No espaço de busca em DPR existem partes invariantes, incluindo a parte fixa de roteamento no FPGA que se mantém constante mas deve ser redefinida devido às limitações de granularidade do DPR, além das codificações inválidas de circuito que podem danificar o FPGA e que não possuem utilidade.

Considerando isto, é possível perceber um ponto muito importante que é consequência da representação compacta de um circuito sob VRC: o tempo de reconfiguração de um circuito através de DPR nunca poderá ser menor que o análogo sob VRC, se ambos implementados optimamente para o mesmo problema.

Além disto, soluções como o DPR possuem como inconveniência a necessidade de adaptação direta do problema à arquitetura do FPGA-destino, enquanto que o VRC é implementado através de HDL *(hardware description language)*, como Verilog ou VHDL *(VHSIC Hardware Description Language)*.

Os VRC, no entanto, tem limitações como o maior consumo de área, devido à necessidade de implementação de todas as funções cabíveis em cada componente, além do uso de numerosos MUX (multiplexador), que aumentam o tempo de propagação dos sinais (SALVADOR et al., 2012).

Sekanina e Friedl (2004) publicaram um trabalho de EHW utilizando VRC para simulação de CLC. Foi adotada uma estrutura com limitações de interconectividade, além de simular LUTs de 2 entradas apenas, viabilizando a operação do VRC a 100MHz. A proposta do VRC não é restrita a uso para CLC e aquele autor não focou os trabalhos posteriores neste tipo de problema específico.

Salvador et al. (2011) apresentou um processador de imagens evolutivo usando DPR, o qual apresentou menor latência de operação e menor uso de área de FPGA quando comparado a VRCs, conforme esperado em tal implementação. No entanto, ele reconheceu que a solução era inferior em termos de tempo de reconfiguração e flexibilidade de granularidade, ao estado-da-arte dos VRC. Ambos os problemas encontrados refletem as dificuldades relativas à reconfiguração direta do FPGA.

3.5 USO DE DPR E VRC EM PESQUISAS RECENTES

Bonilla e Camargo (2011) apresentaram uma plataforma de EHW baseada em FPGA e um processador discreto, para simulação de CLC e EA respectivamente. O autor observou que os trabalhos anteriores utilizavam um microcomputador para o GA e o FPGA para simulações, o que acarretava em perda de velocidade devido a deficiências de comunicação entre estes, em particular a alta latência e baixa velocidade de transmissão. Assim, ao conectar diretamente um microprocessador ao FPGA, foi possível obter um melhor uso do FPGA. Ainda assim, posteriormente, este autor (CABRITA et al., 2013) apresentou um simulador de alta velocidade para CLC, implementado totalmente em software e com resultados comparáveis à implementação FPGA de Bonilla, ressalvando a discrepância entre as frequências de operação FPGA e do processador utilizado entre os dois trabalhos.

Ren et al. (2011) implementou uma solução completa de EHW para CLC utilizando VRC com processador soft-core Nios II. A arquitetura escolhida para o VRC consiste em uma matriz 4x4 de 2-LUTs com severa limitação de interconectividade. O trabalho, no entanto, ficou limitado aos detalhes da implementação e pobre em resultados, limitando-se a publicar um circuito gerado e o consumo de área do FPGA, sem prover SECRT nem o TECC.

Mais recentemente, um trabalho de Wang et al. (2013) incluiu um simulador de CLC implementado em FPGA utilizando DPR, fazendo uso de templates nos quais somente as tabelas LUT são modificadas. De particular interesse é a melhoria na velocidade de reconfiguração, o que é uma deficiência do DPR, quando comparado ao VRC. Wang fez uso de uma estrutura modular, contendo vários simuladores de CLC operando em paralelo, interconectada através de NoC *(network on-chip)* visando alta escalabilidade. Embora obtendo os melhores resultados em simulação de CLC, o trabalho limitou-se a uma parametrização específica e não apresentou resultados refletindo a comportamento real quanto à escalabilidade. Além disso, a estrutura apresentada suporta CLC de geometrias muito restritas, empregando uma matriz 3x4 composta de LUTs de 3 entradas. É perceptível a conveniência de tal geometria considerando as LUT híbridas de 5/6 entradas encontradas na FPGA Virtex-5, utilizada naquela pesquisa. Embora tal arranjo seja muito conveniente para o projeto de um simulador de alta velocidade, é um limitante para a utilidade do circuito-solução, uma vez que prioriza a velocidade do simulador em detrimento da flexibilidade de geometria, esta última não necessariamente a mais adequada para o problema.

O trabalho de Wang não é um caso único, sendo meramente sintomático do que se tornou, nos últimos anos, um padrão frequente em pesquisas envolvendo geração de

CLC através de EA. Tal padrão consiste na prematura simplificação de geometria de CLC, priorizando velocidade de simulação em detrimento da qualidade da solução.

Mesmo em trabalhos anteriores como o de Ke et al. (2010), percebe-se uma estagnação no nível de complexidade dos CLC gerados, quando comparados a trabalhos mais antigos como os de Coello et al. (2000). Considerando as deficiências do estado da arte em EA para geração de CLC, não parece que o problema esteja suficientemente compreendido, a ponto de permitir simplificações agressivas visando exclusivamente velocidade de processamento.

Há, também, uma falta de interesse no desenvolvimento de soluções provendo estruturas de simulação genéricas para CLC, ainda mais as que permitam exploração topológica para fins de pesquisa.

Tal preocupação com simplificação, para fins de velocidade, não está restrita à geometria dos circuitos, havendo trabalhos que o fazem a nível de EA. Um exemplo deste tipo é o trabalho de Cancare et al. (2010), onde foi apresentado um sistema de EHW realizando manipulação direta do bitstream do FPGA, explorando o DPR bidimensional disponível nos FPGA Xilinx Virtex-4 para operações de mutação de alta velocidade, mas com limitações colaterais de nível de granularidade nas operações de mutação, como reflexo das arquitetura específica do componente utilizado.

3.6 TOPOLOGIAS ALTERNATIVAS DE CLC CARENTES DE PESQUISA

Outras arquiteturas menos comuns de CLC continuam sendo investigadas, como o caso de CLCs baseados em MUX 2-1, gerados através de EA (AGUIRRE et al., 1999) com alguma evolução em trabalhos mais recentes (VIJAYAKUMARI et al., 2013). Nesses casos não foi detectado presença de trabalhos com foco na velocidade de simulação de circuitos.

Em paralelo a estes trabalhos, outras pesquisas surgiram a respeito de CLC de forma geral, apontando possíveis novos problemas a serem solucionados através EA. Um caso é o dos CCLC *(cyclic combinational logic circuits)*, atualmente sem pesquisas encontradas que abordem a geração deste tipo de CLC utilizando EA. CCLC, de uma forma geral, é um tema pouco apreciado em pesquisas, e CC (circuito cíclico) tipicamente remete a codificadores/decodificadores, com a própria descrição "circuitos de lógica combinacional cíclicos" podendo parecer contraditória *(oxymoron)* (MALIK, 1993), uma vez que CLCs são circuitos que não apresentam memória dos estados de entrada anteriores.

Tais como demais circuitos cíclicos, os CCLC utilizam realimentação interna de sinais. Intuitivamente, tal realimentação traz como consequência a sensibilidade a diferenças de tempo

de chaveamento dos componentes, assim como as de propagação de um componente a outro. É possível que um circuito tenha comportamento combinacional considerando tempos de atrasos específicos para cada componente ainda que, de outro modo, seja um circuito sequencial. Malik (1993), utiliza como critério a insensibilidade a variações de atrasos nos componentes para que um circuito cíclico seja considerado combinacional, ou seja, um CCLC. Este critério manteve-se o mesmo em trabalhos posteriores de outros autores (RIEDEL; BRUCK, 2003) e (AGARWAL et al., 2005). Tal definição é de extrema importância para EHW, já que permite a avaliação, através de simulação, de um CCLC sem a necessidade da simulação do tempo de propagação do sinal, nem a necessidade de gerar o circuito em sua forma definitiva, se usado DPR. Isso viabiliza o uso de VRCs para simulação de CCLC, bastando que os mesmos suportem realimentação interna.

O trabalho de Mukherjee e Dandapat (2010) aborda CCLC, incluindo dados demonstrando que, para CLCs equivalentes, há menor uso de transistores, área e consumo na implementação cíclica, quando comparado à versão não-cíclica. Para geração de CCLC através de EA é necessário simuladores que permitam realimentação de sinais, o que não é contemplado nos trabalhos atuais.

Este capítulo abordou o estado da arte, descrevendo trabalhos tratando do problema da velocidade de simulação de CLC. Um outro objetivo deste trabalho ao implementar um simulador de CLC, além de alta performance, é a generalidade da solução. O simulador deve ser utilizável até mesmo em topologias de CLC não atualmente contempladas em trabalhos de EA, como o caso dos CCLCs.

O próximo capítulo aborda a arquitetura proposta, composto por uma seção descrevendo a validação por software, assim como a arquitetura em si, implementada em FPGA.

4 ARQUITETURA PROPOSTA

A arquitetura proposta consiste em um simulador de CLC de topologia arbitrária. O projeto visa alta velocidade de simulação e é implementado em VRC sobre FPGA. A estrutura básica consiste em uma matriz de módulos VRC, cada qual composto de um vetor de LUTs virtuais.

Este simulator tem como objetivo o uso para EHW, seja por CGP ou qualquer outro EA. Em paralelo a tais usos, os módulos VRC são genéricos o suficiente para permitir o uso com arquiteturas com demanda para reconfigurabilidade pós-síntese. Cada módulo VRC é um simulador assíncrono de circuito genérico composto por LUTs virtuais, não possuindo dependências de sinais com a lógica de reconfiguração virtual, exceto durante o processo de reconfiguração em si.

Este capítulo está dividido em duas seções:

A seção 4.1 inclui o trabalho anterior deste autor (CABRITA et al., 2013) e consiste na validação em software da arquitetura de uma forma geral e a descrição da forma de interfaceamento do simulador de CLC com o algoritmo evolucionário.

A seção 4.2 trata da arquitetura-foco deste trabalho, e descreve a implementação realizada em FPGA.

4.1 VALIDAÇÃO POR SOFTWARE

Com o objetivo de verificar a viabilidade de uma estrutura modular genérica para simulação de CLC, foi realizada uma implementação em software. Além do simulador em si, é proposta uma estrutura para integração do simulador com o algoritmo genético para fins de geração de CLC.

4.1.1 INTRODUÇÃO

A otimização multi-objetivo de CLC utilizando GA requer um meio de simular os circuitos-candidatos de forma a determinar o fitness de cada indivíduo.

A forma convencional consiste na simulação do circuito para obter o estado de saída para cada possível combinação de entradas, e o incremento de entradas aumenta o número de simulações necessárias exponencialmente. Por exemplo, para uma entrada de 16 bits, um total de $2^{16} = 65536$ simulações são necessárias para avaliar um único circuito-candidato, de uma população que pode ter entre 100-2000 indivíduos, o que resultaria em milhões de simulações por geração. Slowik e Bialko (2008) escreveu sobre o crescimento do número de combinações e consequente tempo de processamento, e citou medidas alternativas para mitigar este problema.

Uma forma de endereçar o problema de performance de simulação seria a criação de um simulador específico para a codificação de circuito utilizada pelo GA. Tal solução traz a inconveniência da perda de flexibilidade do cromossomo, o qual armazena o circuito de forma codificada. Variações estruturais no cromossomo são uma possibilidade, em particular para propósitos de pesquisa, e tais casos criariam a necessidade de reimplementação do próprio simulador e de sua respectiva interface ao GA.

Outra alternativa seria a criação de um simulador de circuitos digitais genérico. Embora tal medida satisfaça as questões de reusabilidade do simulador, o amplo generalismo do mesmo iria penalizar a performance de simulação, assim como trazer maior complexidade de interfaceamento entre o simulador e o GA.

Este trabalho apresenta uma estrutura de simulação de CLC a qual visa endereçar as questões de performance e generalismo. A natureza genérica deste simulador é dentro do escopo de CLC, para fins de uso com GA. O simulador é tão genérico quanto necessário para trabalhos atuais de GA, com o objetivo de geração de CLC, provendo rápida simulação com interfaceamento conveniente ao GA.

Durante a pesquisa foram identificados dois grupos básicos de estruturas de CLC utilizados em GA: o primeiro utiliza uma matriz bidimensional(COELLO et al., 2003) e o outro é baseado em vetor(SLOWIK; BIALKO, 2004).

Neste trabalho foi escolhida uma estrutura baseada em vetor, após considerar os seguintes aspectos:

- Embora a estrutura em matriz seja conveniente para implementações baseadas em FPGA, obtendo vantagem em performance devido ao paralelismo inerente, a mesma impõe

limitações topológicas as quais são detrimentais para a proposta de um simulador genérico. Adicionalmente, tal estrutura não provê benefício, em termos de performance, em implementações baseadas em microprocessadores de propósito geral, os quais endereçam nativamente os dados como vetores.

- Ainda que Vassilev et al. (1999) tenha afirmado que o uso de representação do CLC em matriz apresenta melhor convergência em GA, em contraste com a de vetor, uma estrutura em matriz pode ser facilmente convertida para sua equivalente funcional em vetor, antes da simulação. Isso significa que é possível trabalhar com evolução de CLC organizados em matriz, ainda que o simulador opere nativamente com representação vetorial.

- Não é necessariamente simples a conversão de um CLC estruturado como vetor para matriz.

A continuidade no uso de topologias baseadas em matriz (VIJAYAKUMARI; MYTHILI, 2012) sugere que a escolha o nível de generalização é adequado.

Com o objetivo de viabilizar o uso de um simulador genérico e reutilizável, isolando o mesmo das especificidades da implementação do GA em si, foi definida uma camada funcional intermediária de tradução de cromossomo. Esta camada realiza a conversão da representação do CLC utilizada pelo GA, que é específica, para a representação de CLC genérica utilizada pelo simulador.

4.1.2 A ESTRUTURA MODULAR

De forma a prover um ambiente genérico e reutilizável para o simulador de CLC, uma estrutura modular foi definida, na qual o GA é dividido em blocos funcionais separados. A estrutura modular é mostrada na Fig. 4.

A seção do algoritmo genético, constante na Fig. 4, contém a implementação da parte específica relacionada à otimização, incluindo as operações específicas de GA, a população e a FF *(fitness function)*.

Já a seção de simulação do circuito contém o GCS *(generic circuit simulator)* que é o simulador genérico de CLC, assim como o CTL *(chromosome translation layer)*. A função do CTL é de converter o cromossomo de um indivíduo, que contém internamente uma representação própria do CLC, para GC *(generic chromosome)* que é a representação do mesmo circuito na forma genérica utilizada pelo GCS.

O CTL necessita ser invocado a cada vez que o circuito a ser simulado é modificado,

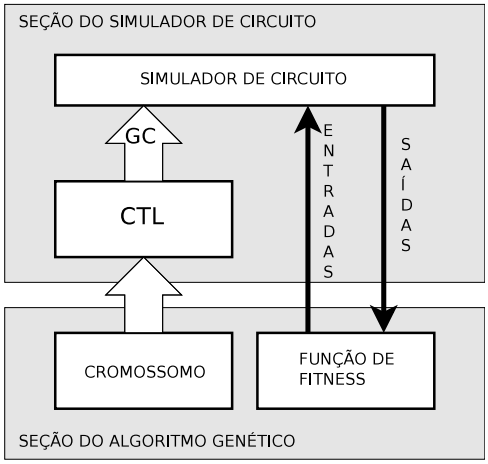


Figura 4: A Estrutura Modular na qual o Simulador de CLC está Localizado

tipicamente uma vez a cada nova avaliação do fitness do indivíduo. Após este procedimento, o circuito poderá ser simulado por tantas vezes quanto necessárias. Considerando que o CTL necessita ser invocado apenas uma vez a cada novo circuito simulado, considerando as múltiplas simulações realizadas, a penalidade de performance é mínima.

A simulação em si é realizada pelo GCS, o qual tem suas entradas estimuladas pelo FF. A forma com que o FF processa ou armazena a saída coletada da simulação não afeta o GCS, uma vez que o FF interage meramente como um cliente do GCS. Colocando de outra forma: o GCS é uma caixa preta para o FF.

4.1.3 O SIMULADOR GENÉRICO DE CIRCUITOS COMBINACIONAIS

O GCS necessita de certos parâmetros fixos, definidos antes do início da operação do GA, os quais são: CIW *(circuit input width)*, CFW *(circuit filling width)* e COW *(circuit output width)*. Cada uma destas reflete a largura dos estados de bits, correspondentes às entradas/saídas dos circuito e das portas lógicas simuladas, e o tamanho do cromossomo genérico, conforme mostrado na Fig. 5.

O CIW é o número total de bits na entrada do circuito, enquanto que o COW é o número de bits de saída. O CFW é escolhido manualmente e define o número de portas lógicas extras, além daquelas atreladas ao COW, presentes no cromossomo. O CFW poderá ser 0.

4.1.3.1 A ESTRUTURA GERAL DO CROMOSSOMO GENÉRICO

O GC é alimentado ao GCS pelo CTL. O generalismo do GC é pelo fato do mesmo superset do cromossomo utilizado pelo GA.

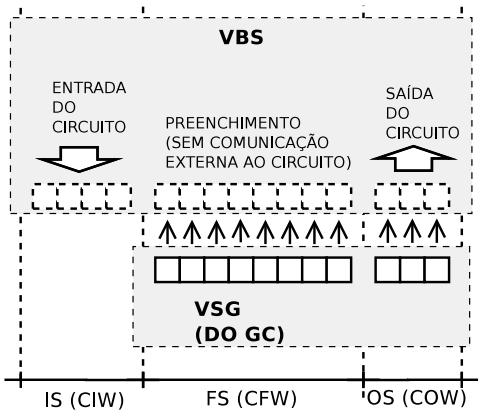


Figura 5: Simulador - Genes e Estados de Bits

O GC é um vetor de genes, sendo que cada gene corresponde a uma porta lógica simulada, esta contendo o tipo de porta e o mapeamento de conexão de cada uma de suas entradas. O mapeamento das entradas e saídas de cada porta lógica é explicado nas seções 4.1.3.2 e 4.1.3.3 respectivamente.

Considerando o valor de CFW como N_{CFW} e o de COW como N_{COW}, o número total de portas lógicas simuladas pelo GCS, N_{total}, é expresso como $N_{total} = N_{CFW} + N_{COW}$, onde $N_{CFW} \geq 0$ e $N_{COW} \geq 1$.

O valor de CIW, N_{CIW}, também deve obedecer a seguinte restrição para ser um circuito válido: $N_{CIW} \geq 1$.

4.1.3.2 GENES E ESTADOS DE BIT

O GCS opera com dois grupos de dados: o VSG *(vector of simulated gates)*, recebido como cromossomo traduzido pelo CTL, e o seu VBS *(vector of bit states)* interno, conforme mostrado na Fig. 5. Ambos os vetores são divididos em três seções: IS *(input section)*, FS *(filling section)* e OS *(output section)*.

O IS não contém portas simuladas, contendo apenas a seção do VBS que representa as entradas do circuito. Estas entradas são providas durante a simulação pelo FF.

Ambos os FS e OS contém portas simuladas e seus estados de bit, sendo que cada um destes estados de bit são atrelados à saída da porta lógica correspondente. O FS e OS diferem entre si na forma em que os estados de bits são tratados: os estados OS são mapeados como saídas do circuito simulado, enquanto que o FS são exclusivamente de uso interno do circuito simulado.

4.1.3.3 SIMULAÇÃO INDIVIDUAL DE PORTA LÓGICA

Cada gene do GC mapeia diretamente a uma única porta simulada. Conforme mostrado na Fig. 6, cada gene, correspondente a uma porta lógica, carrega dois grupos de informação: o mapeamento de conexões de entrada e o tipo de porta lógica, explicado adiante.

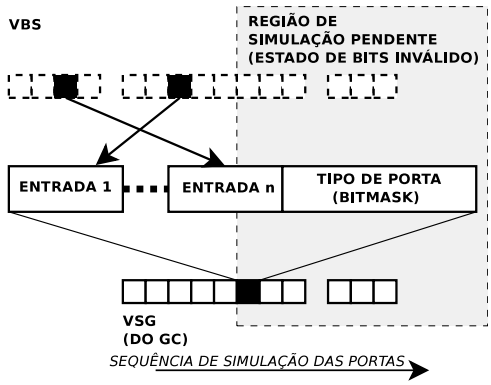


Figura 6: Simulador - Simulação Individual de Porta Lógica

A simulação do circuito é realizada estimulando uma única porta lógica por vez, de forma sequencial da primeira à última. Cada porta coleta suas entradas dentre os estados no VBS, de acordo com o respectivo mapeamento de entradas.

A porta lógica é, então, simulada de acordo com o seu tipo, em um processo explicado na seção 4.1.3.4, e a sua saída é enviada ao seu bit correspondente no VBS. Este processo é repetido até não restarem mais portas a serem simuladas.

Para cada porta simulada na $n^{\underline{a}}$ posição, existe um set limitado de bits válidos como entrada T os quais poderão ser utilizados: $T = n - 1$. A parte restante do VBS é localizada na região de "simulação pendente", a qual contém bits com estados indefinidos (aleatórios). Por questões de performance, o simulador em si não realiza nenhum tipo de verificação de validade de mapeamentos das entradas, deixando ao GA a responsabilidade de prover mapeamentos válidos para cada porta lógica.

4.1.3.4 SIMULAÇÃO DE TIPO DE PORTA LÓGICA

O simulador de portas lógicas imita o comportamento de uma dada operação lógica, como AND, OR, XOR ou outra. Cada porta possui um número fixo de entradas, e todas as portas dentro do circuito simulado devem possuir o mesmo número de entradas. Considerando o número de entradas por porta p, a máscara de bits do tipo de porta (Fig. 7) requer w bits, e o número total de tipos de portas possíveis é s, definido como: $w = 2^p$ e $s = 2^w$, consequentemente

$s = 2^{2^p}$. Entre todos os s tipos de portas, existem operações sem nomenclatura booleana, incluindo tipos de portas que ignoram uma ou mais de suas entradas.

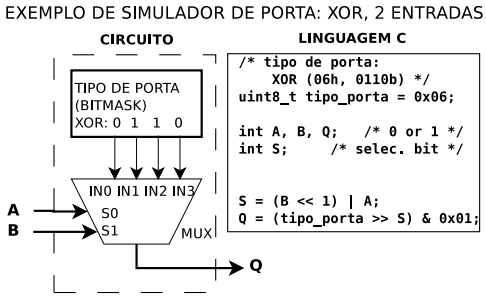


Figura 7: Simulador - Simulação de uma porta XOR, 2 entradas

Esta forma generalizada de simulação de portas lógicas definidas através de máscara de bits, é análoga ao uso de look-up tables (LUT) em FPGA, já utilizado em outros trabalhos (CHEANG et al., 2007), e também implementada utilizando um sistema baseado em multiplexador (HERNANDEZ-AGUIRRE et al., 2000).

Caso o circuito pré-tradução, anterior ao processamento pelo CTL, utilize portas lógicas com número variado de entradas, como AND de 2 entradas e NOT de 1 entrada, o número de entradas a serem simuladas p deve ser de acordo com a porta lógica com o maior número de entradas. Em tais casos o CTL deve mapear os tipos de portas para outras que simulem portas com menos entradas que p, tornando inativas as portas em excesso.

Especificamente na questão de GA, Coello e Aguirre (2002) apontou não haver vantagens no uso de uma máscara binária para definir os tipos de portas lógicas (utilizando operações genéticas sobre os bits individuais), sobre uma codificação baseada em alfabeto (tratando o tipo de porta lógica como uma partícula indivisível). Embora o GCS trabalhe com máscara binária, isto não afeta a codificação interna utilizada pelo GA, uma vez que através do CTL é possível mapear a forma de representação de porta lógica para outra.

O simulador de portas consiste em um multiplexador com o fim de simular uma lookup table. O comportamento está exemplificado na Fig. 7, onde a simulação de uma porta XOR de 2 entradas é mostrado, em diagrama esquemático e em linguagem de programação C.

4.1.3.5 ENTRADAS E SAÍDAS DO CIRCUITO SIMULADO

Conforme explicado anteriormente, as entradas do circuito simulado estão mapeadas diretamente nos primeiros bits correspondentes do VBS, que correspondem ao IS.

Uma vez que todas as portas foram simuladas, a saída do circuito simulado já está

presente nos últimos COW bits do VBS, e reflete o estado dos bits situados na OS (Fig. 5).

4.1.4 TESTES DE PERFORMANCE DO SIMULADOR

4.1.4.1 METODOLOGIA

O simulador de CLC baseado no sistema acima descrito foi implementado em linguagem de programação C, sendo verificado o seu correto funcionamento antes dos testes de performance.

Testes de performance em lote foram realizados com tamanhos variados de circuitos e demais parâmetros. O processamento realizado nos testes não incluiu processamento de GA, uma vez que isso iria interferir nas estatísticas de performance.

A métrica de performance foi definida como o número de simulações completadas do circuito para um único valor de entrada, por segundo. Assumindo um GA sem cache de fitness/cromossomo, onde todas as possíveis combinações binárias de entrada devem ser verificadas, considerando o número de CIW N_{CIW} e o valor de simulações por segundo S_{perf}; a performance de simulação para um único indivíduo I_{perf} e para uma única geração G_{perf} com uma população de P_{total} pode ser calculada conforme a Eq. 1 e a Eq. 2.

$$I_{perf} = \frac{S_{perf}}{(2)^{N_{CIW}}} \tag{1}$$

$$G_{perf} = \frac{I_{perf}}{P_{total}} \tag{2}$$

Os resultados mostrados na Fig. 8 e na Fig. 9 incluem o overhead (custo extra de processamento) do CTL, já que o mesmo é um requisito para o sistema proposto, ainda que não seja parte do processamento do GA. Os resultados mostrados na Fig. 10 não incluem o overhead do CTL como os dois anteriores, neste caso os cromossomos foram diretamente mapeados no simulador, pela fato da implementação do CTL suportar somente portas lógicas com 2 entradas.

Os circuitos simulados foram gerados aleatoriamente a cada rodada, sendo o mesmo para todos os indivíduos da mesma rodada. No total, 4 rodadas foram incluídas nos gráficos dos resultados. Embora todos os indivíduos fossem os mesmos, a cada rodada o CTL foi invocado por indivíduo, de forma a contabilizar o overhead de processamento do mesmo, como iria ocorrer caso o simulador estivesse integrado a um GA.

A simulação foi realizada por indivído, cada qual com cobertura de todas as possíveis

combinações binárias das entradas.

4.1.4.2 HARDWARE E SOFTWARE UTILIZADOS NOS TESTES

O hardware sobre o qual os testes foram realizados é um computador de mesa com processador Intel Core i7 2600K de 3.4 GHz, com 16 GB de memória DDR3-1333 em dois canais de memória. Foi utilizado o sistema operacional Debian Linux 6.0.6.

Esta implementação por software não suporta multithreading. Como consequência, somente um núcleo do processador foi utilizado, de um total de 4 presentes no processador.

4.1.4.3 RESULTADOS

A curva de performance para valores de CFW com valores fixos de bits de entrada/saída (cada qual setado em 12) e 2 entradas por porta lógica é mostrada na Fig. 8. A curva inicia com CFW em 1, com resultado de 45M simulações/s e termina com mais de 5M simulações/s para um CFW de 100.

Consta na Fig. 8, assim como em gráficos subsequentes, 4 linhas com valores muito próximos. Estas linhas se tratam das 4 rodadas de testes, ou seja, os 4 testes que foram realizados separadamente e sobrepostos no mesmo gráfico. O simulador foi executado em um sistema operacional multitarefa que possui processos paralelos de baixa carga executando em paralelo, o que cria pequenas variações de performance do simulador ao longo do tempo. Com o objetivo de discriminar estas pequenas variações de performance do simulador em si, estes múltiplos testes foram sobrepostos no mesmo gráfico.

O impacto na velocidade de simulação com o incremento de número de entradas do circuito é mostrado na Fig. 9. O menor número de entradas do circuito resulta em menos simulações realizadas por indivído, aumentando a proporção do overhead por processamento do CTL, o que explica a redução de performance à medida que o número de entradas diminui. A partir de um certo número de bits de entrada a curva de performance, novamente, volta a diminuir. Isto ocorre devido à pressão do cache do processador consequente dos acessos do buffer, de grande tamanho, onde são armazenados os resultados da simulação.

O simulador suporta um número variado de entradas por porta lógica, este número refletindo em todas as portas lógicas do circuito. A degradação de performance é mostrada na Fig. 10.

A variação no número de bits de saída do circuito, desde que o número total de portas

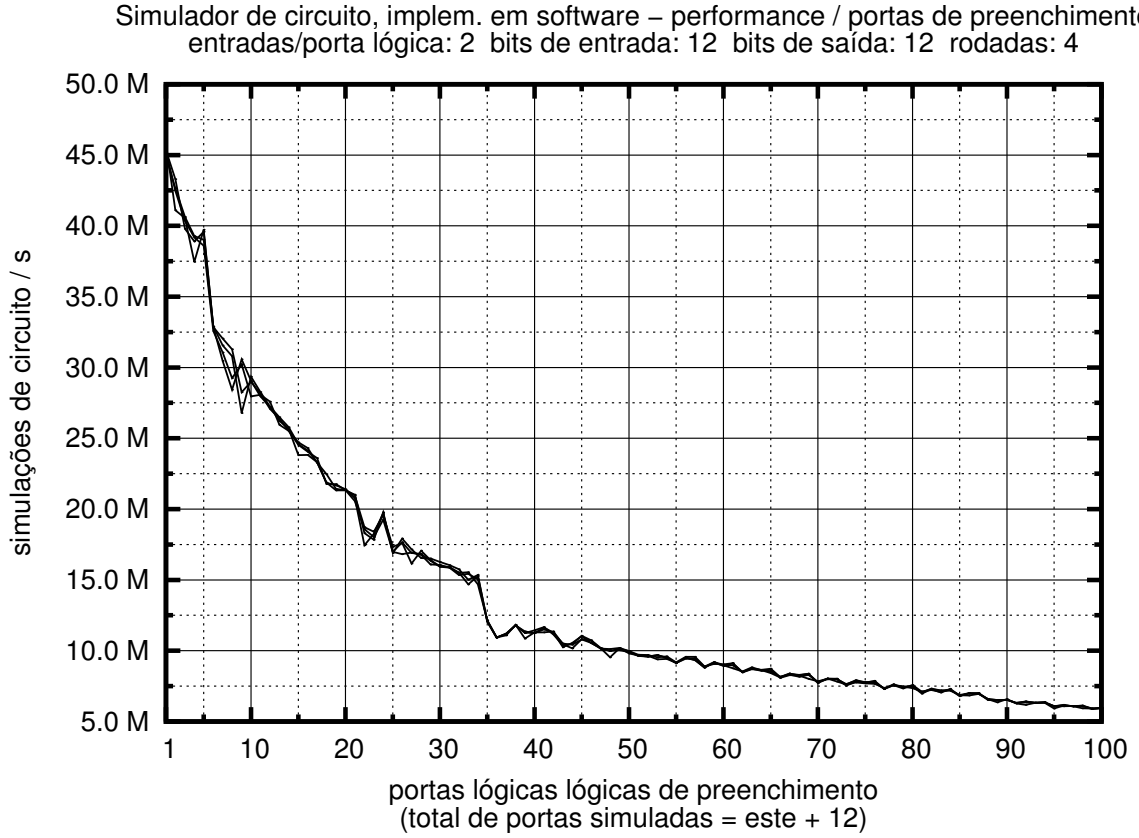


Figura 8: Performance de simulação por CFW

lógicas se mantenha o mesmo, não varia a performance de simulação. Por esta razão, o gráfico correspondente não foi incluido.

4.1.4.4 COMPARAÇÃO DE PERFORMANCE COM OUTROS TRABALHOS

Benchmarks de simulação de circuitos não são comumente encontrados em trabalhos de geração de CLC através de GA. Os dados encontrados em outros trabalho tiveram de ser extraídos indiretamente.

A performance do trabalho de Bonilla e Camargo (2011) na sua implementação SIE baseada em FPGA, considerando: 12 bits de entrada, 512 indivíduos e 100 gerações, resulta em um total de 209.715.200 simulações. Embora não esteja claro quantas portas lógicas foram efetivamente simuladas, não há dados suficientes para obter o impacto do incremento de portas com relação à performance. Utilizando 1 nó SIE, o tempo de execução foi de 5 segundos, consequentemente a performance obtida foi de 41.943.040 simulações/s. Este número inclui o overhead do GA, o qual operou por 100 gerações.

Conforme mostrado na Fig. 8, para circuitos de 12 bits de entrada, o simulador

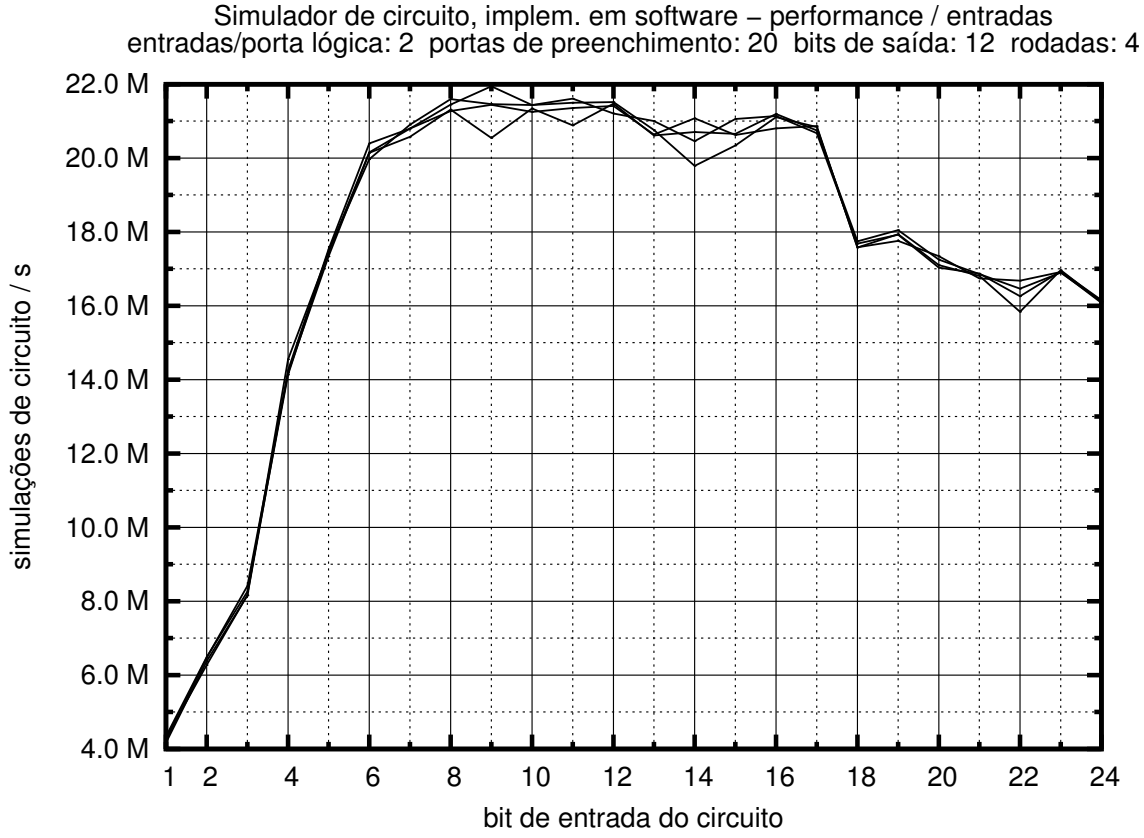


Figura 9: Performance de simulação por CIW

deste trabalho, implementado inteiramente em software, obteve mais de 20M simulações/s para $CFW = 20$, e 10M simulações/s para $CFW = 50$. Os resultados mostram que a performance de simulação em software mantém-se viável para circuitos com número total de portas lógicas $CFW + COW$ maior que 50.

4.1.5 CONCLUSÃO

Foi apresentada uma estrutura concisa para simulação de alta performance de CLC, com níveis de performance encorajadores, em particular considerando que a implementação foi realizada totalmente em software, sem o uso de FPGA.

Circuitos com $CIW \leq 3$ claramente sofrem penalidade na performance durante a simulação, isto devido a overhead do CTL. Tais casos não são relevantes na prática, considerando a simplicidade de tais circuitos, resultado em rápida convergência através de GA. Circuitos com $CIW \geq 8$ não sofrem de impacto discernível pelo uso do CTL, se considerado apenas o processamento do simulador.

A performance de simulação, no entanto, é negativamente afetada por pressão de

Figura 10: Performance de simulação por número de entradas por porta lógica

cache do processador, a qual aumenta à medida que o CIW incrementa, devido ao consequente aumento de tamanho do buffer, em memória, que armazena as saídas do circuito para posterior análise. É esperado que a pressão de cache aumente em uma implementacão multithreaded, devido ao compartilhamento de cache entre múltiplos cores do processador. Este problema poderá ser mitigado através de gerência explícita de cache, sendo esta uma possível extensão desta implementação.

A estrutura do simulador pode ser extendida de forma a prover estatísticas como: contagem de portas úteis, uso de transistores, latência do circuito e outros. Tais extensões podem ser implementadas utilizando o GC oriundo do CTL, evitando implementações específicas a codificações de cromossomos e garantindo a reusabilidade do código.

Para pesquisas futuras, pode mostrar-se viável uma estrutura semelhante de simulação para operações com valores numéricos, permitindo rápida avaliação de expressões matemáticas geradas por GA.

4.2 DESCRIÇÃO DA ARQUITETURA

Este trabalho consiste em uma arquitetura para simulação de CLC, otimizada para uso com EHW. Esta é estritamente uma infraestrutura para simulação, a ser usada para a avaliação do correto funcionamento de múltiplos circuitos-candidato. A simulação do CLC, em si, é o passo com maior consumo de tempo durante o processo de cálculo do fitness em implementações de EA e um alvo maior de optimização.

Considerando o foco sendo estritamente o simulador em si, foi necessário a implementação de um circuito auxiliar para efetuar testes com a arquitetura proposta. Tanto os simuladores quanto o circuito de testes foram implementados integralmente em FPGA. A arquitetura testada inclui um computador externo, o qual efetua o upload das descrições dos circuitos a serem simulados, coletando posteriormente o resultado dos testes. O computador é conectado ao FPGA através de um link RS232, escolhido por conveniência de prototipagem. A baixa velocidade serial do link não é relevante neste trabalho, já que não afeta a velocidade de simulação.

4.2.1 IMPLEMENTAÇÃO GLOBAL

A arquitetura é composta por múltiplos VRCM *(virtual reconfigurable circuit modules)* e todos operando em paralelo e de forma sincronizada. De outro modo, cada VRCM é um VRC independente. Conforme mostrado na Fig. 11, um VRCM possui duas seções funcionais: CCS *(circuit configurator section)* e a VCS *(virtual circuit section)*. Uma vez que o circuito foi configurado, o VCS opera analogamente a um CLC ordinário sintetizado diretamente em FPGA, exceto apenas pelos tempos de resposta mais lentos.

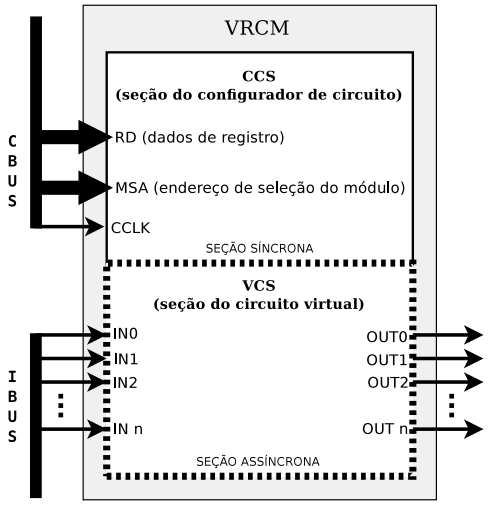


Figura 11: Apresentação externa do VRCM.

O diagrama de implementação global é mostrado na Fig. 12. A arquitetura global inclui múltiplos VRCMs, os quais de número total especificado antes da síntese em FPGA. O MC *(main controller)* configura os VRCMs através do CBUS *(configuration bus)* e define o SCLK *(simulation clock)*, que é a frequência de simulação dos VRCMs. O MC também está conectado ao FE *(fitness evaluator)*, que é o circuito que controla o ciclo de testes de CLC, além de coletar os resultados da avaliação de cada CLC avaliado.

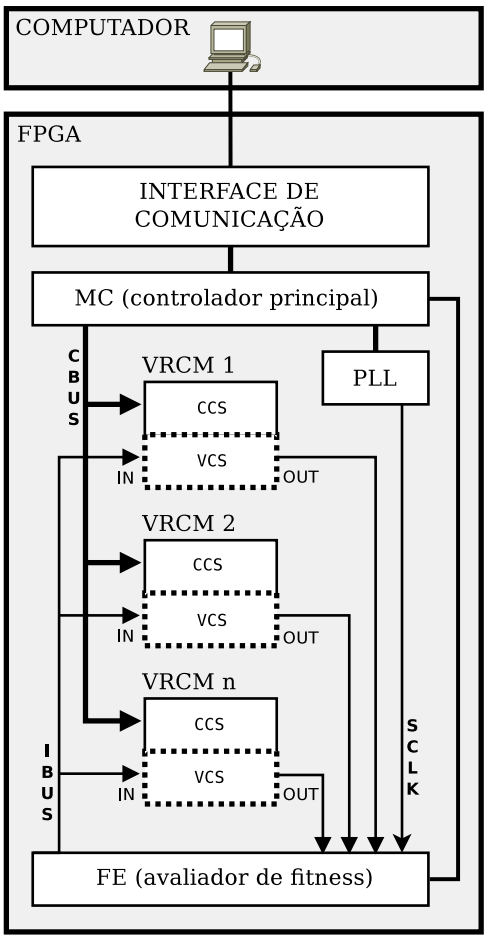


Figura 12: Diagrama global.

O FE gera um ciclo completo de testes combinacionais. Durante cada ciclo, cada possível combinação de entradas é testada. O FE alimenta as entradas dos VRCM através do IBUS *(input bus)*, consequentemente todos os VRCMs compartilham dos mesmos sinais de entradas de forma síncrona. O FE coleta todas as saídas dos VRCMs em paralelo, já que todos os circuitos-candidatos são avaliados simultaneamente. Cada estado combinacional no IBUS é mantido por exatamente um clock do SCLK, consequentemente o SCLK deve acomodar o tempo de propagação interno dos VRCMs.

Neste trabalho o FE é usado estritamente para efetuar testes de correto funcionamento dos VRCMs, sob certas frequências SCLK, e levantar a máxima frequência de operação para o

conjunto de VRCMs. Para uma implementação completa de EA, é necessário que o FE inclua todo o circuito necessário para o cálculo do fitness, o que é fora do escopo deste projeto.

4.2.2 IMPLEMENTAÇÃO DO VRCM

A apresentação externa de um VRCM individual é mostrado na Fig. 11. Cada VRCM é um VRC completo e independente, com seu próprio circuito de reconfiguração. Embora os VRCMs estejam sendo usandos para simulação para fins de EHW, o mesmo poderá ser implementado como parte de um circuito funcional, substituindo partes que, de outro modo, seriam estáticas após a síntese no FPGA.

O CBUS transporta todos os sinais necessários para a reconfiguração do circuito, através do qual o CCS recebe a descrição do circuito a ser virtualizado.

O VCS é a parte do VRC apresentando o circuito funcional, o qual simula o CLC configurado. A simulação é realizada de forma assíncrona, consequentemente as saídas respondem às mudancas das entradas tão rápido quanto permitido pelo tempo de propagação interno do VRCM. O VCS em si não possui dependência a qualquer fonte de clock, uma vez que é um circuito assíncrono. Na implementação global, o VCS é parte do domínio de clock SCLK, devido à interconexão com o FE. O FE é um circuito síncrono a SCLK, conforme mostrado na Fig. 12.

A estrutura interna do VRCM é mostrada na Fig. 13. Cada VRCM é composto por múltiplos VLUT *(virtual look-up table)*. Cada VLUT possui seu próprio registro, o qual mantém uma máscara de LUT e os endereços de mapeamento das entradas do LUT.

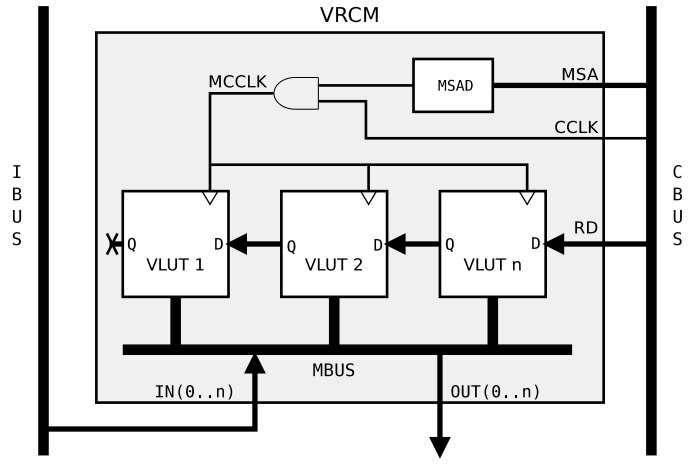


Figura 13: Diagrama interno do VRCM.

O CCS recebe a descrição do circuito na forma de múltiplos RD *(register data)*,

recebendo um a cada CCLK *(configuration clock)*. Cada RD mapeia integralmente o registro interno de um único VLUT.

O MSA *(module selection address)* faz a função de seleção de chip *(chip select)* durante a configuração. Cada módulo responde a um único endereço MSA, o qual é definido na síntese do circuito. O MSAD *(module selection address decoder)* sinaliza 1 quanto o MSA coincide com o código do VRCM local.

Em cada VRCM, o set local de registradores VLUT é organizado em um pipeline, sincronizado pelo MCCLK *(module configuration clock)*. O uso de pipeline impede a reconfiguração de VLUTs individuais, consequentemente a granularidade mínima para reconfiguração parcial é um VRCM integralmente.

Cada VRCM contém um MBUS *(local module bus)*, o qual é utilizado para transporte interno dos sinais do circuito, incluindo sinais de um VLUT a outro. Parte dos sinais no MBUS são conectados às entradas do VRCM, ou às saídas deste último.

4.2.3 IMPLEMENTAÇÃO DA VLUT

Cada VLUT contém uma LUT virtual reconfigurável, assim como a configuração dos mapeamentos de entrada. Uma VLUT se comporta como uma LUT, com circuito de roteamento agregado em suas entradas.

Conforme mostrado na Fig. 14, cada VLUT possui um único registro, o qual mantém a VLUT MASK *(LUT bit mask)*, assim como o seletor dos sinais a serem usados como entradas da LUT. O registro é atualizado a partir da entrada de registro (D) em MCCLK. A saída do registro (Q) é utilizada para pipelining do RD, conforme descrito anteriormente e mostrado na Fig. 13.

A mesma saída Q é utilizada internamente para configurar o comportamento da VLUT. Q é dividida em seções, cada qual usada para selecionar os sinais específicos do MBUS como entrada (INPUT 1..n), assim como para a definição da VLUT MASK.

Cada VLUT pode ter duas ou mais entradas, conforme especificado antes da síntese do circuito. Este parâmetro se aplica a todas as VLUTs do VRCM, assim como a todos os VRCMs.

Cada um dos sinais nas entradas de uma VLUT é escolhido através de um MUX, a partir de todos os sinais disponíveis no MBUS. Os sinais selecionados são utilizados para o seletor do VLUT MASK, o que determina a saída da VLUT. A saída da VLUT, por sua vez, é mapeada para uma posição fixa e única dentre os sinais do MBUS.

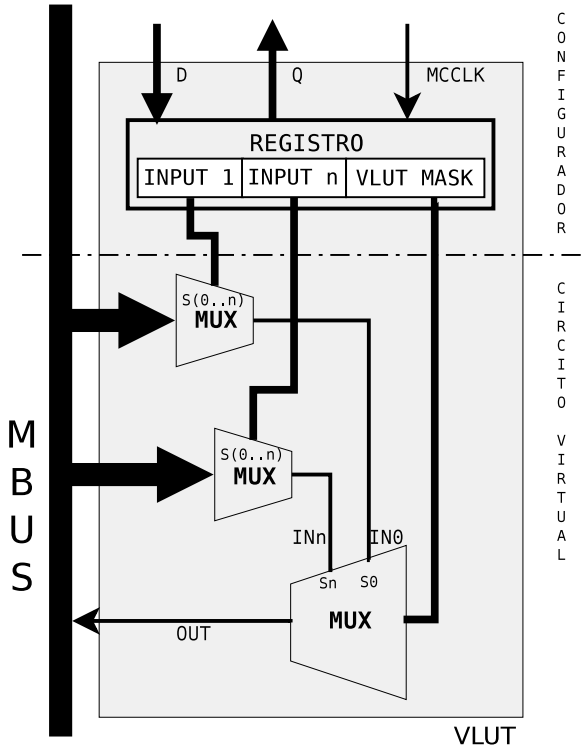


Figura 14: Diagrama interno da VLUT

Conforme mostrado na Fig. 15, cada sinal no MBUS tem uma única origem fixa, definida antes da síntese do circuito. Cada um dos sinais do IBUS, assim como as saídas de cada VLUT, são conectados a posições únicas e fixas no MBUS. O efeito deste arranjo é que as saídas não podem ser reconfiguradas. Considerando que cada saída é conectada a um único sinal MBUS, tal estrutura impede a conexão entre duas ou mais saídas. Isto torna o VRCM eletricamente seguro para todas as possíveis configurações de circuito.

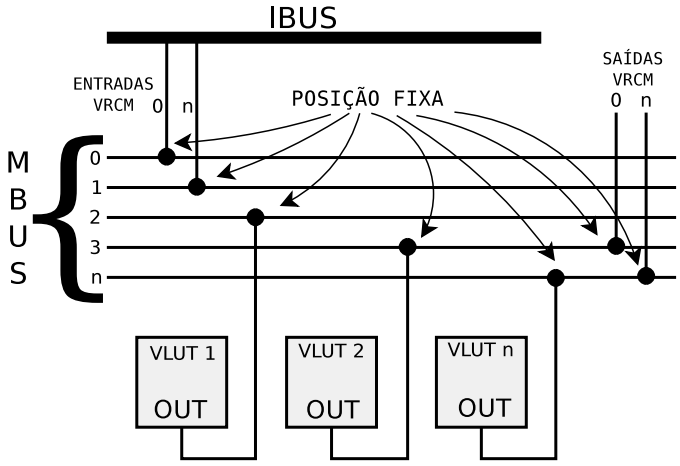


Figura 15: Saídas das VLUTs: mapeamento fixo.

A saída do VRCMs é simplesmente um mapeamento dos últimos *n* sinais do MBUS,

refletindo os *n* bits de saída do VRCM. Isto foi adotado como uma convenção, embora nada impeça o uso de quaisquer outros sinais do MBUS como bits de saída. Havendo necessidade para tal, o mapeamento das saídas do VRCM pode se tornar indiretamente reconfigurável se as VLUTs correspondentes forem definidas como buffers não-inversores, e utilizados para remapeamento dentre os demais sinais do MBUS. Tal técnica evitaria a necessidade de resíntese do simulador para mudanças arbitrárias no mapeamento das saídas com os sinais do MBUS.

A operação do mapeamento de entradas da VLUT é mostrada na Fig. 16, onde as entradas VLUT (IN 0..n) correspondem a IN 0..n da Fig. 14. Uma VLUT pode utilizar qualquer dos sinais disponíveis no MBUS, e qualquer sinal poderá ser utilizado mais de uma vez, ou mesmo nunca utilizado. Qualquer combinação é válida, embora não necessariamente gere circuitos considerados úteis.

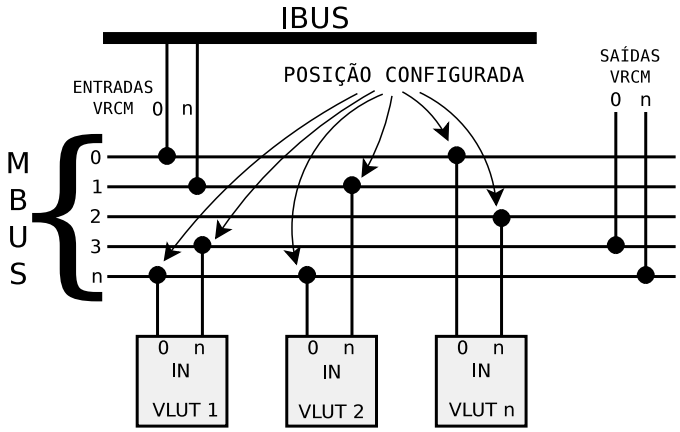


Figura 16: Entradas das VLUTs: mapeamento variável.

A estrutura de simulação de CLC proposta oferece modularidade e generalidade, esta última visando independência de algoritmo evolucionário a ser utilizado. O capítulo a seguir descreve a metodologia dos testes realizados de forma a determinar o comportamento dessa estrutura quando implementado sobre FPGA.

5 METODOLOGIA DOS TESTES

Os testes consistem no levantamento de performance dos VRCM considerando parâmetros diversos de:

1. Número de VLUTs por VRCM.

2. Número de VRCMs no FPGA utilizado.

Os resultados, análise de dados com gráficos, e comparativos com outros trabalhos estão presentes no capítulo 6.

A arquitetura proposta foi implementada em um FPGA Altera Cyclone IV EP4CE115F29C7N, o qual possui 114,480 LE *(logic elements)*, cada qual contendo um LUT nativo de 4 entradas. Os circuitos foram sintetizados utilizando o Quartus II 64-Bit Version 13.0.1 Build 232.

Nenhum dos circuitos gerados durante os testes foram individualmente otimizados, consequentemente os resultados reportados não são necessariamente os melhores alcançáveis e algumas configurações de circuitos podem apresentar melhor performance que casos mais simples.

De forma a realizar os testes, levantando tanto a máxima frequência de operação e estabilidade do circuito, um TC *(test circuit)* fixo é implementado tanto como VRC, para upload em cada VRCM, e uma instância hardcoded e embutida no FE (mostrado na Fig. 12). O FE, repetidamente e simultaneamente, estimula tanto o TC interno como os programados em todos os VRCMs, pelo tempo mínimo de 300 segundos. O FE também compara a saída do seu TC interno, que é a saída de referência, com as saídas de todos os VRCMs. Se qualquer saída de qualquer VRCM, em qualquer momento, não for idêntica ao esperado, o VRCM correspondente é persistentemente marcado como falho.

Os testes visam determinar qual a máxima frequência de clock em que o conjunto dos VRCMs não apresentam erros em funcionamento estável. Foi definido como tempo de teste 300 segundos que, se atingido sem erros, o circuito é considerado estável. O tempo de 300 segundos

foi estimado após resultados de testes preliminares: em todos os casos que houve erros, os mesmos ocorreram antes de 30 segundos, sendo que os demais casos operaram sem erros até o desligamento, após funcionamento contínuo por um tempo mínimo de 1200 segundos.

Os testes para levantamento de máxima frequência de clock em operação estável foram realizados através do seguinte procedimento:

1. O FPGA é carregado com o circuito contendo os VRCMs a serem testados.
2. O circuito de testes, um somandor, é carregado nos VRCMs através de um cabo serial.
3. É definido, através de chaves na placa contendo o FPGA, o divisor de frequência do clock original de 100MHz para que forneça um clock de baixa frequência, com o qual é esperado que o circuito opere de forma estável.
4. O divisor de frequência é modificado manualmente de forma a, progressivamente, aumentar o clock de operação, até atingida a frequência em que ocorrem erros de forma imediata.
5. O divisor de frequência é incrementado em 1 (um), o que resulta na diminuição do clock de saída. É realizado um reset do status de erro, através de um botão na placa do FPGA. Após isso, é aguardado 300 segundos.
6. Se não ocorreram erros, o circuito é considerado estável sob aquele clock. Do contrário, repete-se o passo acima.
7. O divisor de frequência é decrementado em 1 (um), o que resulta no aumento do clock de saída. É aguardado até 300 segundos para que ocorra o erro, que é esperado. Este passo visa confirmar o clock máximo de operação estável.
8. Confirmado o erro, o circuito é considerado estável sob o clock máximo de valor imediatamente inferior ao atual, do contrário o experimento é repetido do início.

O estímulo gerado pelo FE para todos os TCs consiste na saída de um contador, intercalado com seu complemento. Assim, para um TC com 5 bits de entrada, a sequência binária é 00000, 11111, 00001, 11110, 00010 e assim por diante. Este padrão foi escolhido com o objetivo de manter todos os bits de entrada em alta frequência de transição, além de cobrir todas as possíveis combinações de entradas.

O TC escolhido é um somador completo, composto de somadores completos de 1 bit encadeados. Este foi escolhido considerando o trânsito dos sinais carry ao longo de todo o

circuito. Com este padrão de propagação, as saídas do VRCM se tornam estáveis somente após todas as VLUTs, uma após a outra, se estabilizarem. Este foi considerado um circuito de pior caso para CLC não-cíclicos, em termos de tempo de propagação de sinal.

Embora os testes tenham sido realizados com parâmetros variados de VRCM, alguns parâmetros foram restritos com o objetivo de simplificar as tabelas de resultados. O registro VLUT, mostrado na Fig. 14, possui largura de $w_{register}$ bits. Tamanhos fixos de $w_{register}$ foram utilizados, de acordo com o número de entradas por VLUT n_{input}, conforme mostrado na Tab. 1. O tamanho $w_{register}$ varia de acordo com: n_{input}, a largura da máscara LUT w_{mask}, e a largura de cada mapeamento de entrada w_{input}. Os parâmetros são calculados conforme as Eqs.3 e 4.

Tabela 1: Larguras dos registros VLUT usadas durante os benchmarks

n_{input}	w_{input} (bits)	w_{mask} (bits)	$w_{register}$ (bits)
2	6	4	16
3	5	8	23
4	6	16	40
5	6	32	62
6	5	64	94
7	6	128	170

$$w_{mask} = 2^{n_{input}} \tag{3}$$

$$w_{register} = n_{input} \times w_{input} + w_{mask} \tag{4}$$

Considerada esta metodologia e $w_{register}$ como parâmetro fixo, este último a depender do valor de n_{input}, foram realizados testes de performance e consumo de área de FPGA com variações nos demais parâmetros e os resultados apresentados no capítulo a seguir.

6 RESULTADOS

Os resultados são mostrados na Tab. 2, com parâmetros variados para um único VRCM, e na Tab. 3 com variável número de VRCMs. Os parâmetros são o número de entradas por VLUT n_{input}, o número de VLUTs que compõe cada VRCM $\frac{VLUT}{VRCM}$, e o total número de VRCMs $VRCM_{total}$, também incluindo o número de VRCMs que falharam $VRCM_{fail}$, se houverem e entre parênteses. $VRCM_{fail}$ é o número de VRCMs que não operaram 100% corretamente durante o tempo de teste.

SCLK, conforme descrito anteriormente, é a frequência sob a qual o circuito virtualizado sobre VRCM opera. SCLK é gerado a partir de um clock de 100MHz dividido por um valor inteiro, e possui DC *(duty cycle)* de 50%. SCLK reflete o total número de simulações por segundo realizadas pelos VRCMs.

Além disso, considerando que o FE estimula os VRCMs durante a borda de subida do clock, avaliando as saídas na borda de descida, o tempo efetivo de propagação de sinal no VRCM t_{VRCM} não é maior que a metade do período de SCLK. O valor máximo de t_{VRCM} pode ser calculado através da Eq. 5, onde DC é o duty cycle (percentual). Isto representa uma possibilidade interessante para otimização, visando maior performance: se for usado um SCLK com DC assimétrico, os circuitos virtualizados sobre VRCM poderão operar sob clock ainda maior.

$$t_{VRCM} \leq \frac{1}{SCLK} \times \frac{DC}{100} \tag{5}$$

Considerando SCLK como um valor de benchmark, a performance total obtida através da operação paralela de múltiplos VRCMs Σ_{SCLK} é calculada conforme mostrado na Eq. 6.

$$\Sigma_{SCLK} = (VRCM_{total} - VRCM_{fail}) \times SCLK \tag{6}$$

Os dados de consumo de recursos da FPGA para a implementação global (esta mostrada na Fig. 12), também estão presentes: LE sendo o total de LEs, dos quais CF é o

total de funções combinacionais, e *LR* os registros lógicos dedicados.

No caso do SCLK provido na Tab. 2, o mesmo se refere ao clock máximo possível sem ocorrência de erros causados por mau funcionamento do VRCM, conforme procedimento de teste descrito no capítulo 5.

Os SCLK providos na Tab. 3 são determinados diferentemente e refletem o clock para maior Σ_{SCLK} possível, mesmo que isso acarrete em falha de parte dos VRCMs. Neste caso, que é o de múltiplos VRCMs operando em paralelo, o critério de medição foi alterado durante os experimentos, após ser percebido que a busca por 0% de falha em todos os VRCMs não necessariamente resulta na melhor performance total Σ_{SCLK}. Considerando que é possível utilizar apenas os VRCMs que não apresentarem falhas, ao custo de desperdício de área de FPGA, foi adotado como critério o clock que forneça o maior Σ_{SCLK}, calculado conforme a Eq. 6.

Conforme pode ser visto na Tab. 2, a performance SCLK decrementa com o incremento de qualquer um dos fatores: a relação $\frac{VLUTs}{VRCM}$ e n_{input}. Esta perda de performance era prevista. No caso do n_{input} isto reflete o tamanho da VLUT, em termos de consumo de células do FPGA. A outra razão é por aumentar o número total de conexões virtuais no VRCM devido ao maior número de entradas da VLUT, o que aumenta a pressão por vias de roteamento interno no FPGA. No caso da relação $\frac{VLUTs}{VRCM}$, ocorre o aumento do número total de sinais possíveis para cada entrada de VLUT, refletindo diretamente no aumento de entradas, e consequente complexidade, de cada MUX conectado a cada uma das entradas entrada das VLUTs (Fig. 14 e 16). Conforme o número de entradas de cada MUX, há um nível de cascateamento das LUTs nativas do FPGA, já que cada MUX é composto por uma ou várias LUTs nativas, o que aumenta o tempo de resposta destes MUXes.

A Tab. 3 é similar à Tab. 2, porém inclui múltiplos VRCMs ao invés de apenas um. Conforme pode ser visto na tabela, o total de VRCMs é outro fator que, quando incrementado, causa impacto negativo na performance SCLK. É pertinente lembrar que os VRCMs operam em paralelo, mas o conjunto de VRCMs opera de forma síncrona (Fig. 12). Os VRCMs ocupam, cada um, regiões distintas do FPGA, mas todos com uma região comum para chegada do sinal, o que acarreta em diferenças de roteamento entre cada VRCM. As diferenças de roteamento causam diferenças no tempo de propagação do sinal até a região comum (FE) que, por tratar todos os VRCMs de forma síncrona, fica limitada ao VRCM com maior latência. À medida em que aumenta o número de VRCMs, há o aumento do número de sinais, assim como o aumento do raio de distância máxima entre os VRCMs até o FE, causando diferenças na distância de roteamento e consequente aumento de tempo de propagação do pior caso.

Um outro fator que causa diferença de tempo de resposta entre os VRCMs são as variações de características elétricas ao longo da área do semicondutor do FPGA em si, causadas por imperfeições no processo de fabricação (GUAN et al., 2013). Existem linhas de pesquisa dedicadas à síntese de circuitos, em particular *placement* e roteamento, buscando minimizar o impacto das imperfeições do semicondutor e maximizar a performance, até fazendo uso de mapeamento individual do FPGA, levantando características de performance de cada região do semicondutor (YU et al., 2011). A estratégia de sacrificar um número de VRCMs de forma a obter a maior performance do conjunto, ainda que grosseira, é uma forma de aumentar o aproveitamento do FPGA, considerando as características individuais do componente. Esta estratégia tem a conveniência de não necessitar de síntese específica ao specimen FPGA, pois os testes dos VRCMs são realizados pós-síntese. Após tais testes, é possível determinar quais VRCMs podem ser desconsiderados.

Tabela 2: Performance pelo número de VLUTs, único VRCM

n_{input} (entradas por VLUT)	$\frac{VLUTs}{VRCM}$	$VRCM_{total}$	SCLK (MHz)	$w_{register}$ (bits)	LE	CF	LR
2	10	1	33.33	16	533	459	274
2	15	1	20.00	16	792	724	369
2	20	1	14.29	16	1168	1090	443
2	25	1	11.11	16	1647	1557	569
2	30	1	9.09	16	2223	2123	653
2	35	1	9.09	16	2897	2719	737
2	40	1	6.67	16	3672	3474	821
2	45	1	5.88	16	4368	4150	905
2	50	1	5.26	16	4824	4586	989
3	6	1	33.33	23	517	435	285
3	8	1	33.33	23	655	556	356
3	10	1	33.33	23	813	727	406
3	12	1	33.33	23	1031	937	456
3	14	1	20.00	23	1281	1179	506
3	16	1	20.00	23	1472	1362	556
3	18	1	14.29	23	1620	1502	606
3	20	1	14.29	23	1768	1642	656
4	6	1	33.33	40	647	524	369
4	8	1	20.00	40	837	686	473
4	10	1	20.00	40	1058	919	549
4	12	1	14.29	40	1357	1202	625
4	16	1	11.11	40	2034	1843	845
4	20	1	9.09	40	3024	2801	1013
4	22	1	7.69	40	3473	3234	1097
4	26	1	7.69	40	4678	4307	1265
4	30	1	5.26	40	5953	5534	1433
4	40	1	4.35	40	7992	7453	1853
5	6	1	33.33	62	843	647	509
5	10	1	20.00	62	1403	1163	780
5	12	1	14.29	62	1807	1535	898
5	14	1	11.11	62	2263	1959	1016
5	16	1	9.09	62	2709	2368	1219
5	18	1	9.09	62	3351	2973	1347
5	20	1	7.69	62	3989	3584	1475
5	30	1	5.26	62	7799	7089	2115
5	40	1	4.35	62	10489	9569	2755
6	4	1	33.33	94	802	564	582
6	6	1	20.00	94	1164	831	761
6	8	1	20.00	94	1532	1117	995
6	10	1	14.29	94	1955	1518	1187
6	12	1	11.11	94	2495	1994	1379
6	14	1	9.09	94	3103	2538	1571
6	16	1	9.09	94	3595	2966	1763
6	18	1	7.69	94	3994	3301	1955
6	20	1	7.69	94	4400	3643	2147
7	4	1	33.33	170	1215	780	922
7	6	1	20.00	170	1743	1145	1237
7	8	1	20.00	170	2305	1559	1616
7	10	1	14.29	170	2923	2098	1946

Tabela 3: Performance pelo número de VRCMs

n_{input} (entradas por VLUT)	$\frac{VLUTs}{VRCM}$	$VRCM_{total}$	$VRCM_{fail}$	SCLK (MHz)	Σ_{SCLK} (MHz)	$w_{register}$ (bits)	LE	CF	LR
2	10	72		33.33	2400.0	16	23395	19777	12060
3	10	8		33.33	266.7	23	4926	4401	2415
3	12	9		20.00	180.0	23	7429	6771	3120
3	10	80	1	20.00	1580.0	23	47268	42210	23079
3	12	80	1	20.00	1580.0	23	64271	58573	26763
4	10	4		20.00	80.0	40	3561	3076	1839
4	10	64		14.29	914.3	40	53602	46160	27639
4	10	80	8	14.29	1028.6	40	66950	57652	34519
5	10	64		14.29	914.3	62	76249	62343	42423
5	10	80	3	14.29	1100.0	62	95261	77883	52999
6	10	32	5	14.29	385.7	94	55896	42630	34295
6	10	60	5	11.11	611.1	94	104620	79762	64199
6	10	64		11.11	711.1	94	111582	85062	68471
7	10	20	3	11.11	188.9	170	54460	38402	36659
7	10	40	7	11.11	366.7	170	108703	76605	73199

6.1 ANÁLISE DE GRÁFICOS

Com o objetivo de realizar uma análise por inspeção visual dos dados coletados durante os experimentos, foi utilizado os dados da Tab. 2 para a geração de gráficos. Não foram gerados gráficos com a Tab. 3, a qual apresenta resultado para múltiplos VRCMs, já que a mesma conta com um número reduzido de entradas. O tempo reduzido para a conclusão desta pesquisa tornou inviável extender os experimentos com múltiplos VRCMs, já que que são circuitos de demandam muito tempo para síntese.

A partir dos dados da Tab. 2 foi gerado o gráfico constante na Fig. 17, o qual demonstra a relação de performance SCLK com o número de VLUTs. É perceptível o impacto negativo do número de VLUTs para a frequência máxima de operação SCLK. O número de entradas por VLUT n_{input} reflete o tipo de LUT virtual, assim uma VLUT com $n_{input} = 2$ equivale a uma 2-LUT, $n_{input} = 3$ a 3-LUT e assim por diante.

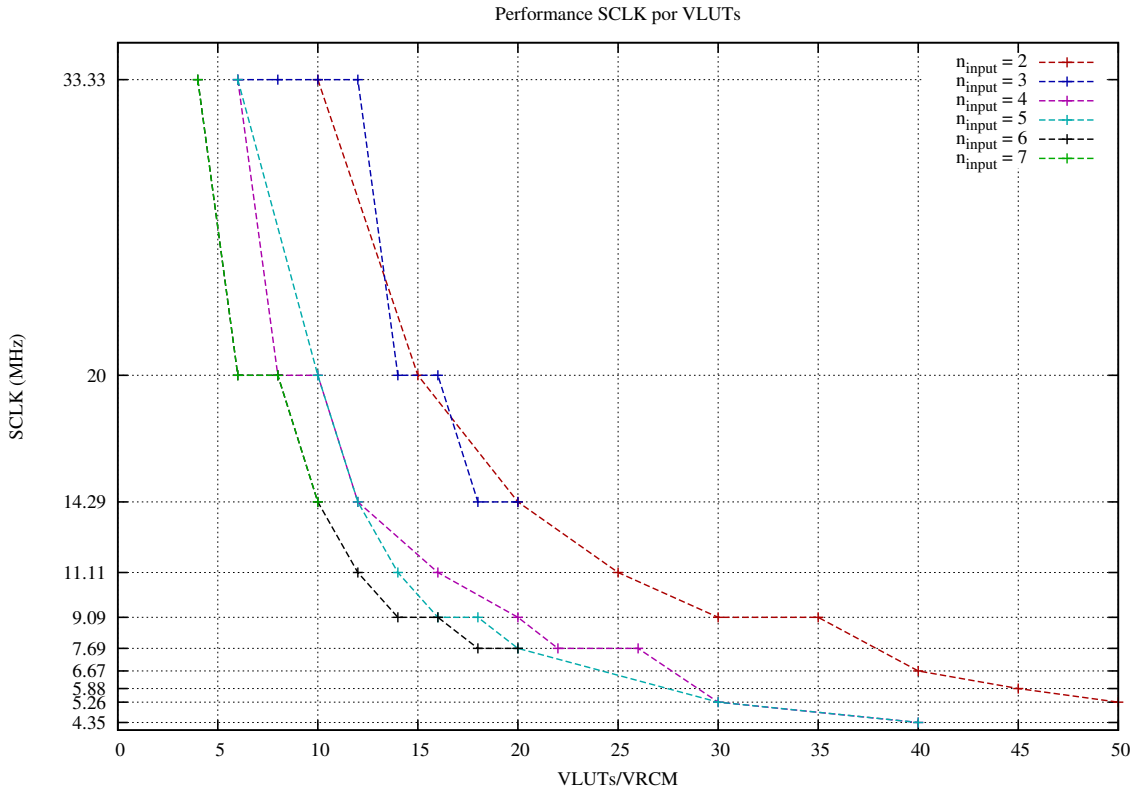


Figura 17: Performance SCLK por VLUTs, único VRCM

É também perceptível o que parece ser um padrão de proximidade de performance quando $n_{input} = 2$ e $n_{input} = 3$, se comparado aos demais valores n_{input}. No entanto, o número de amostras para $n_{input} = 3$ é muito limitado para assumir tal padrão, sendo a razão desta limitação é o fato de, para $n_{input} = 3$, $w_{input} = 5$ bits (Tab. 1), o que limita o tamanho máximo

de endereçamento de VLUTs. Para gerar mais dados, foi necessário refazer o experimento alterando para $w_{input} = 6$ bits, sendo os resultados apresentados na Tab. 4, onde $n_{input} = 3(a)$ representam os novos dados gerados.

Tabela 4: Performance pelo número de VLUTs, único VRCM, $n_{input} = 3(a)$ $w_{input} = 6bits$

n_{input} (entradas por VLUT)	$\frac{VLUTs}{VRCM}$	$VRCM_{total}$	SCLK (MHz)	$w_{register}$ (bits)	LE	CF	LR
3	4	1	50.00	26	393	334	242
3	6	1	33.33	26	517	435	285
3	8	1	33.33	26	656	558	356
3	10	1	33.33	26	814	729	406
3	12	1	20.00	26	1032	939	456
3	14	1	20.00	26	1282	1181	506
3	16	1	20.00	26	1526	1413	607
3	18	1	14.29	26	1879	1731	663
3	20	1	14.29	26	2258	2129	719
3	22	1	14.29	26	2587	2450	775
3	24	1	11.11	26	3080	2866	831
3	26	1	9.09	26	3470	3242	887
3	30	1	9.09	26	4411	4155	999
3	34	1	9.09	26	5059	4775	1111
3	36	1	7.69	26	5339	5041	1167
3	38	1	7.69	26	5613	5301	1223
3	40	1	6.67	26	5894	5568	1279
3	42	1	7.69	26	6171	5831	1335

Utilizando os dados da Tab. 2 e substituíndo as entradas onde $n_{input} = 3$ pelos dados constantes na Tab. 4, foi gerado o gráfico da Fig. 18. Neste gráfico é mais claramente perceptível a proximidade de performance SCLK para $n_{input} = 2$ e $n_{input} = 3$, havendo uma distância entre estes e os demais valores de n_{input}. É possível que este efeito seja decorrente de maior afinidade de síntese das VLUT com as 4-LUT nativas do FPGA, encorajando um teste em FPGAs com arquitetura baseada em 6-LUT, o que não foi possível nesta pesquisa por indisponibilidade de tal hardware.

Um outro fato percebido na Fig. 18, na série $n_{input} = 3$, é uma anomalia no valor para 40 VLUTs que cai para 6,67 MHz, retornando a 7,69 MHz com 42 VLUTs. Este resultado sugere uma manifestação das limitações do algoritmo heurístico do fitter/roteador que, mantidos os demais parâmetros de síntese utilizados nos outros testes, obteve solução pior para um caso ligeiramente mais simples.

O consumo de LEs do FPGA pelo total de VLUTs, efetivamente a relação de consumo de área do FPGA pelo tamanho do VRCM, foi colhido dos dados da Tab. 2 e Tab. 4, combinando os dados $n_{input} = 3$ e $n_{input} = 3(a)$ em um único gráfico, com o resultado mostrado

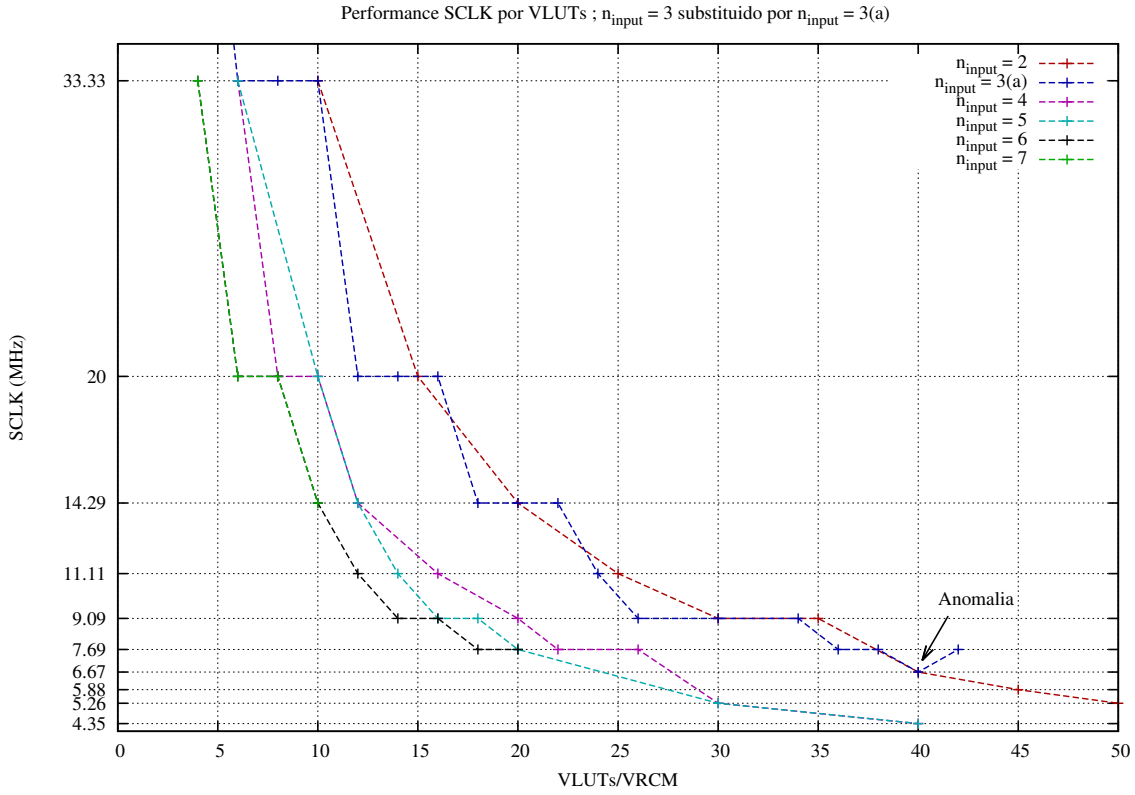


Figura 18: Performance SCLK por VLUTs, único VRCM ; $n_{input} = 3$ substituído por $n_{input} = 3(a)$

na Fig. 19, apresentando comportamento esperado.

6.2 COMPARAÇÃO COM OUTROS TRABALHOS

Foram escolhidos dois trabalhos para comparação com o estado da arte em performance de simulação de CLC para fins de EHW, utilizando FPGA: o trabalho de Bonilla e Camargo (2011) por utilizar a técnica de VRC, comparado na seção 6.2.1, e o trabalho de Wang et al. (2013) por utilizar DPR, comparado na seção 6.2.2.

Um terceiro trabalho, deste autor (CABRITA et al., 2013), foi incluído nas Tab. 5 e 7, sendo escolhido por se tratar de um simulador de CLC implementado inteiramente em software, já inserido como parte deste trabalho na seção 4.1.

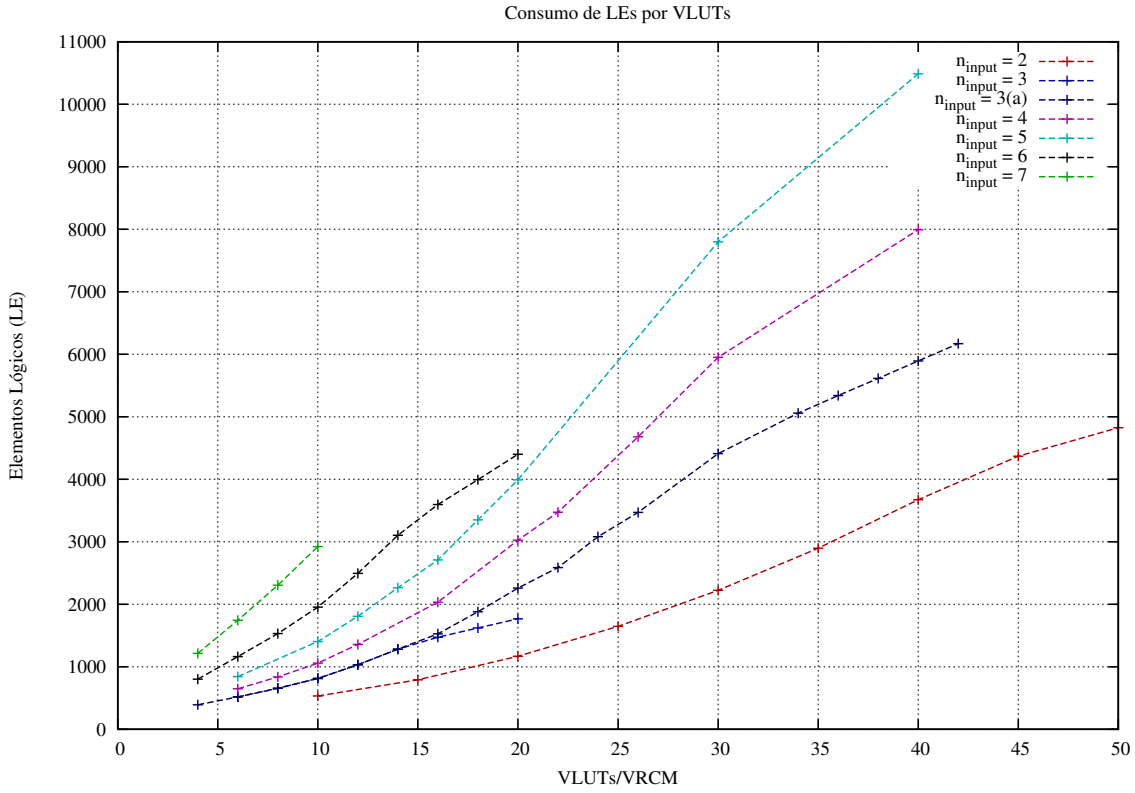


Figura 19: Consumo de LEs do FPGA por total de VLUTs, único VRCM

6.2.1 BONILLA

Bonilla e Camargo (2011) apresentou uma plataforma de EHW baseada em FPGA e um processador externo, para simulação de CLC e EA respectivamente. Foi utilizado VRC na parte da implementação realizada em FPGA. A codificação utilizada para os CLC é a de árvore, diferentemente da codificação utilizada neste trabalho.

Bonilla utilizou um FPGA para acelerar o processo de avaliação dos CLC, sendo este FPGA conectado diretamente a um SoC *(system on a chip)* que, por sua vez, realiza as demais tarefas de GP. Este conjunto de FPGA com SoC foi chamado de SIE. A comparação é restrita à performance de avaliação de CLC, realizado pelos respectivos VRC.

A estrutura utilizada por Bonilla possui limitações dos CLC que pode representar: os CLCs podem ser compostos somente por operandos booleanos, ou 2-LUT, não suportando o uso de LUTs de tamanhos arbitrários, e o fato dos circuitos possuírem apenas um bit de saída, que seria o "topo" da árvore de operandos booleanos. Outra limitação da arquitetura é o fato da mesma implementar apenas um VRC por FPGA, sendo que os 6 nós SIE citados naquele trabalho totalizam 6 FPGAs e refletem os 6 VRCs disponíveis.

O autor não provê detalhes suficientes da implementação do VRC. Os benchmarks

não fornecem o número de LUTs virtuais que os CLC testados possuem, mas é informado que o cromossomo é composto por células *(cells)* cada qual composta de 3 portas lógicas. Considerando que o maior CLC apresentado no artigo, gerado pelo algoritmo, totaliza 7 portas lógicas, foi estimado que os CLC dos benchmarks utilizam um VRC realizando a operação de 9 portas lógicas no total.

O trabalho não provê benchmarks para comparação direta de performance do VRC, razão pela qual o valor foi calculado a partir do gráfico "Response time for 12 variables", linha "SIE 512 individuals", para 1 nó SIE. Considerando que o gráfico reflete o tempo gasto de $t = 5$ segundos, para $g = 100$ gerações, de $i = 512$ indivíduos para VRC de $b = 12$ bits de entrada (chamado de "variables"), a performance do VRC SIE_{perf} é de 41,943040 MHz, conforme calculado pela Eq. 7.

$$SIE_{perf} = \frac{2^b \cdot g \cdot i}{t} \tag{7}$$

O comparativo entre o trabalho de Bonilla e este consta na Tab. 5. É possível perceber uma vantagem de performance do SIE, com 41,943 MHz, sobre este trabalho, com VRCM a 33,33 MHz. No entanto, conforme descrito anteriormente, cada módulo SIE opera apenas com um VRC, assim cada FPGA possui apenas um VRC operante. Diferentemente do SIE, neste trabalho cada FPGA opera com múltiplos VRCs (VRCM) simultaneamente. No teste comparativo foram utilizados 72 VRCMs no total, com performance conjunta de 2400 MHz, que é muito superior ao SIE e demonstra um melhor aproveitamento do FPGA.

Tabela 5: Comparação entre os trabalhos, CLC composto por 2-LUT

descrição	Bonilla	Cabrita 2013	este trabalho
FPGA/CPU da implementação	Xilinx XC3S500E	Intel i7 2600K	Altera Cyclone IV
técnica	VRC	software	VRC
topologia	árvore 2-LUT	cartesiana n-LUT	arbitrária
entradas por LUT	2	2	2
LUTs por circuito	9 (estimado)	13	10
t_{config} (ciclos de clock)	(sem informação)	(software)	12
módulos TEHW/VRCM/CPU em operação paralela	1	1	72
performance individual de TEHW/VRCM/CPU (frequência de clock do CLC, MHz)	**41.943**	**45**	**33.33**
performance agregada paralela de módulos TEHW/VRCM (soma da frequência de clock, MHz)	**41.943**	**45**	**2400**

6.2.2 WANG

Wang et al. (2013) implementou uma solução completa da EHW em FPGA, com a qual será comparada apenas a parte pertinente ao simulador de CLC. Naquele trabalho foi utilizado uma NoC para interconectar os TEHW *(target EHW module)*, os quais são os análogos aos VRCMs do presente trabalho. Os TEHW são baseados em DPR, ao invés de VRC, consequentemente o circuito opera em velocidades nativas do FPGA. Uma comparação geral entre a arquitetura do corrente trabalho e a proposta por Wang é mostrada na Tab. 6.

Wang apresentou um TEHW estruturado como uma matrix 4x3 de 3-LUT, com interconectividade limitada a LUTs de colunas vizinhas, apenas. É claramente otimizada para LUTs híbridas de 5/6 entradas dos FPGA Virtex-5, ao custo de severas limitações de geometria do circuito.

Tabela 6: Comparação geral entre os trabalhos

descrição	Wang	este trabalho
técnica / portabilidade	DPR (específico a fabricante/família)	VRC (Qualquer FPGA)
topologia	Matriz 2D com interconectividade. limitada	Topologia arbitrária, graças à interconectividade completa.
limitações de geometria do circuito	Menciona apenas circuitos com 3-LUT que caibam em matrix 4x3	a) Todas as VLUTs do VRCM devem ter o mesmo n_{input} b) Área de FPGA
paralelismo escalável	Não demonstrado com mais de 9 unidades TEHW	Impacto na performance, dependendo de parâmetros (ver Tab. 3)

Também são comparados os benchmarks entre o simulador apresentado por Wang e um desde trabalho e sintetizado com parâmetros equivalentes ou superiores ao referido trabalho. Os benchmarks são mostrados na Tab. 7, onde t_{config} é o tempo, em ciclos de clock, necessário para configurar um único TEHW/VRCM com 12 LUTs/VLUTs. Ambos os simuladores possuem 12 LUTs/VLUTs. Enquanto Wang suporta entradas e saídas com no máximo 4 bits cada, os VRCMs não possuem tal limitação, sendo que os VRCMs utilizados na comparação possuem características superiores: entradas de 13 bits e saídas de 7 bits. Os VRCMs também suportam topologias arbitrárias de circuito, enquanto que os módulos TEHW não possuem tal flexibilidade.

Embora os módulos TEHW tenham maior performance de simulação que os VRCM individualmente, não foi demonstrado a escalabilidade dos TEHW além de 9 unidades, enquanto foi demonstrado que os VRCM são capazes de escalar até 79 unidades operacionais,

para simulação paralela de múltiplos CLCs. Com estas considerações, a performance agregada dos VRCMs supera a dos TEHWs, conforme Tab. 7: 900 MHz (Wang) versus 1580 MHz (este trabalho).

Tabela 7: Comparação entre os trabalhos, CLC composto por 3-LUT

descrição	Cabrita 2013	Wang	este trabalho
FPGA/CPU da implementação	Intel i7 2600K	Xilinx Virtex-5	Altera Cyclone IV
técnica	software	DPR	VRC
topologia	cartesiana n-LUT	cartesiana 3-LUT	arbitrária
entradas por LUT	3	3	3
LUTs por circuito	52	12	12
t_{config} (ciclos de clock)	(software)	24	14
módulos TEHW/VRCM/CPU em operação paralela	1	9	79
performance individual de TEHW/VRCM/CPU (frequência de clock do CLC, MHz)	**9.5**	**100**	**20**
performance agregada paralela de módulos TEHW/VRCM (soma da frequência de clock, MHz)	**9.5**	**900**	**1580**

7 CONCLUSÃO

Este trabalho apresentou uma arquitetura de alta performance para simulação de circuitos de lógica combinacional de topologia arbitrária, implementada em FPGA e utilizando a técnica VRC. A proposta desta arquitetura é a sua utilização para geração de circuitos de lógica combinacional através de algoritmos evolucionários, tais como Algoritmos Genéticos, Programação Genética e outros. Esta arquitetura é parte da seção encarregada do cálculo do Fitness, para a qual é necessária um simulador de circuitos de alta velocidade, este utilizado para a avaliação da qualidade da saída do circuito-candidato através de comparativo com a tabela-verdade esperada.

Slowik e Bialko (2008) já haviam reconhecido como um problema a questão de velocidade de simulação do circuito de lógica combinacional para cálculo de Fitness, e nos últimos anos foram propostas soluções para aumento de performance, em particular as baseadas em FPGA. Os trabalhos que propuseram um simulador de alta perfomance o fizeram, quase todos, como uma solução completa de geração de circuitos de lógica combinacional, incluindo já o algoritmo evolucionário, *ainda que o problema de performance de simulação do circuito seja separado ao do algoritmo evolucionário*. Salvo em artigos publicados em revista, os autores, ao tratar dos dois problemas conjuntamente, precisam comportar suas pesquisas nos limites de páginas de artigos para congresso, tipicamente entre 4 a 6 páginas, sendo obrigados a simplificar o trabalho a ponto de haver diversas omissões, como no caso dos trabalhos com os quais este foi comparado: Bonilla e Camargo (2011), e de Wang et al. (2013).

Ainda assim, a insistência de diversos autores em tratar ambos os problemas em um mesmo trabalho acabou transformando essa forma em um padrão de fato para tais pesquisas, a tal ponto que durante a apresentação de um trabalho deste autor (CABRITA et al., 2013) foi questionado a falta de implementação do algoritmo evolucionário.

A questão específica de simulação de CLC é o foco deste trabalho, sendo que os resultados demonstram que a arquitetura proposta excede a performance agregada de Bonilla, e mesmo à implementação baseada em DPR de Wang, esta última limitada a FPGAs com tal

suporte. É particularmente importante o fato deste trabalho, baseado em VRC, ter ultrapassado a performance agregada de outro baseado em DPR, demonstrando a viabilidade de uso de VRC para uso de simulação de CLC, mesmo se comparado com DPR que tem como principal vantagem a performance. Considere-se ainda as desvantagens do DPR como a dificuldade de implementação, disponibilidade restrita a certos modelos de FPGA, além da não-portabilidade entre FPGAs de famílias distintas.

Embora existam outros trabalhos propondo simuladores de CLC implementados em FPGA, já citados neste trabalho, os mesmos não fornecem resultados suficientes para comparação de performance do simulador em si, razão pela qual a performance deste trabalho foi comparada apenas com os trabalhos de Wang e Bonilla.

Nos experimentos foi percebido um padrão de performance similar de 2-VLUT e 3-VLUT (análogos VRC a 2-LUT e 3-LUT), se comparado aos casos de 4-VLUT e superiores. Isso sugere uma limitação da arquitetura nativa 4-LUT do FPGA e instiga a investigação de ganho de performance em FPGAs de arquiteturas baseadas em 6-LUT. Adicionalmente, é esperado um ganho de performance ao utilizar 6-LUT nativas devido ao uso de MUXes de grande número de entradas pelas VLUTs. Elementos lógicos com 6-LUT são tipicamente encontrados em FPGAs recentes topo-de-linha, embora já existam linhas de baixo custo da Xilinx com suporte nativo a 6-LUT, como a Spartan-6 (XILINX, 2010), Artix-7 (XILINX, 2014a) e a SoC Zynq-7000 (XILINX, 2014b), que podem ser mais indicadas para VRC que o Altera Cyclone IV, limitado a 4-LUT e utilizado nesta pesquisa.

No capítulo 6 foi identificado uma possibilidade de otimização visando aumento de performance dos VRCMs, através de alteração do duty-cycle do clock, considerando o relativamente longo tempo de propagação do sinal, através dos VRCMs, em contraste com o rápido tempo de resposta do circuito avaliador de resultado, o FE.

É interessante observar o que os resultados sugerem ser um caso prático das limitações do algoritmo heurístico do software de síntese FPGA, em que um circuito obtém performance inferior a uma variante ligeiramente mais complexa do mesmo. No caso em questão, observado na seção 6.1, tal anomalia de performance foi observada na Fig. 18 entre VRCMs similares, diferindo em apenas um parâmetro.

Um ponto negligenciado nos trabalhos envolvendo simuladores de CLC é a questão de flexibilidade estrutural do simulador de forma a suportar topologias diversas de circuito. O trabalho de Bonilla trabalha apenas com circuitos compostos por operadores booleanos, ou 2-LUT, fornecendo apenas um sinal (bit) de saída do circuito. O trabalho de Wang aceita apenas circuitos compostos por 3-LUT, dispostos em uma matriz de tamanho fixo e com

interconectividade limitada às LUTs de colunas vizinhas. Os limites impostos por ambas estas arquiteturas não apenas prejudicam a utilidade destes simuladores para fins de pesquisa, mas sequer permitem CLCs baseados em 4-LUT, de uso comum em arquiteturas FPGA de baixo custo atualmente disponíveis no mercado.

Em contraste a outros trabalhos, a arquitetura proposta nesta pesquisa é estruturalmente parametrizável, suportando variações diversas de: a) tamanho de LUTs, b) total de LUTs por circuito, c) número de entradas e saídas do circuito, d) livre conectividade entre as LUTs, permitindo topologia arbitrária, incluindo realimentação.

O suporte a realimentação é uma provisão, a ser explorada em futura pesquisa, para viabilizar a simulação de circuitos *cíclicos* de lógica combinacional. Dentre os trabalhos envolvendo algoritmos evolucionários para geração de CLC, não foram encontrados simuladores de CLC com suporte a realimentação. Também não foram encontrados trabalhos envolvendo geração de CCLC através de algoritmos evolucionários de uma forma geral, o que demonstra carência de pesquisas neste tema.

Ainda que a arquitetura proposta suporte circuitos cíclicos, testes CCLCs não foram incluídos nesta pesquisa. Há questões adicionais que devem ser tratadas, considerando o uso do simulador para geração de circuitos através de algoritmos evolucionários. Uma questão, que envolve ambos CLC e CCLC, é o tempo de propagação dos sinais, a partir da entrada, até que a saída se torne estável. No caso de CLC não-cíclicos é possível simplesmente adotar uma codificação pré-VRCM que não permita um circuito com realimentação, e utilizar o tempo de pior circuito-caso para definição do clock de simulação. Já no caso dos CCLC, como é necessário permitir circuitos realimentados, a estratégia do pior-caso torna-se inviável pois o circuito-candidato pode ser: a) instável, não sendo um CCLC, b) estável, funcionalmente combinacional mas dependente das temporizações específicas do VRC utilizado, não cumprindo com a definição de CCLC de Malik (1993), c) estável e rigorosamente um CCLC, porém com longo tempo para estabilização. Embora as questões específicas de CCLC não façam parte do escopo deste trabalho, para futura pesquisa pode ser considerado utilizar o simulador para um teste preliminar do circuito-candidato, adotando uma janela de tempo máximo para estabilização do circuito e descartando circuitos que se mantenham instáveis após aquele tempo.

Embora a simulação de CCLC para fins de avaliação de circuito-candidato ainda possua questões a serem trabalhadas, a simulação de CLC não-cíclicos, que é a forma comum de circuitos de lógica combinacional, não parece ter impeditivos para geração de circuitos relativamenteo complexos através de algoritmos evolucionários.

Realizando uma rápida revisão da evolução da complexidade dos CLC gerados por algoritmos evolucionários, em pesquisas realizadas nos últimos anos, começando com o trabalho de Coello et al. (2000), que inclui como caso mais complexo um CLC composto por 7 portas lógicas, limitado a 4 bits de entrada e 3 de saída. No que toca os algoritmos evolucionários em si, para geração de CLC, não parece ter havido avanços significativos de complexidade dos circuitos gerados desde os trabalhos de Slowik e Bialko (2006), que conseguiu gerar CLC de 4 bits de entrada e 3 de saída, codificado em uma matriz de 5x5 elementos booleanos e minimizando o uso de transistores. Anos após, Ke et al. (2010) trabalhou com CLC codificados em uma matriz de 4x4 elementos booleanos, limitado a 4 bits de entrada e saída cada, evoluindo um multiplicador 2x2 bits. Bonilla e Camargo (2011) evoluiu comparadores de 2 bits, também construídos com operadores booleanos, sendo que naquele trabalho o autor concede ter dificuldades em gerar circuitos mais complexos. Em um trabalho mais recente, Wang et al. (2013) trabalhou com uma matriz de 4x3 composta por 3-LUT, evoluindo circuitos de 3 bits de entrada e 4 de saída, de "funções randômicas".

É perceptível nos trabalhos acima, que refletem desde trabalhos mais antigos até o estado da arte, o que pode ser visto como uma estagnação nas pesquisas envolvendo algoritmos evolucionários para geração de CLC. Considere-se o nível modesto de complexidade dos circuitos gerados pelo estado da arte, apresentando circuitos não mais complexos que os gerados por trabalhos mais antigos. Isto torna questionável a insistência de certos autores em tratar este problema em conjunto com um simulador de CLC visando alta performance, ao invés de tratar as duas questões separadamente.

A arquitetura de simulação de CLC proposta neste trabalho, embora motivada para uso com algoritmos evolucionários visando a geração de circuitos, não possui dependência em tal técnica e sua utilidade não está restrita a tal fim. É pertinente enfatizar que cada VRCM é um circuito virtual composto por LUTs, com interconectividade livre entre entradas e saídas destas LUTs, e apresentado externamente como um componente. Em casos aplicáveis, é possível utilizar o VRCM em substituição ao DPR, para uso em FPGAs que não possuam tal recurso, seja para reconfiguração dinâmica de codificadores/decodificadores, seja para lógica combinacional complexa que necessite ser reconfigurada dinamicamente. As limitações de granularidade de área reconfigurável, assim como a falta de portabilidade do DPR podem tornar o uso dos VRCMs vantajoso em certos casos específicos.

REFERÊNCIAS

AGARWAL, V. et al. An efficient combinationality check technique for the synthesis of cyclic combinational circuits. In: **Design Automation Conference, 2005. Proceedings of the ASP-DAC 2005. Asia and South Pacific**. [S.l.: s.n.], 2005. v. 1, p. 212–215 Vol. 1.

AGUIRRE, A.; COELLO, C. A. C.; BUCKLES, B. A genetic programming approach to logic function synthesis by means of multiplexers. In: **Evolvable Hardware, 1999. Proceedings of the First NASA/DoD Workshop on**. [S.l.: s.n.], 1999. p. 46–53.

ALTERA. **Increasing Design Functionality with Partial and Dynamic Reconfiguration in 28-nm FPGAs**. July 2010. Disponível em: <http://www.altera.com/literature/wp/wp-01137-stxv-dynamic-partial-reconfig.pdf>.

ALTERA. **Dynamic Reconfiguration in Stratix IV Devices**. January 2014. Disponível em: <http://www.altera.com/literature/hb/stratix-iv/stx4_siv52005.pdf>.

BONILLA, C.; CAMARGO, C. Low cost platform for evolvable-based boolean synthesis. In: **Circuits and Systems (LASCAS), 2011 IEEE Second Latin American Symposium on**. [S.l.: s.n.], 2011. p. 1–4.

CABRITA, D.; GODOY, W.; LOPES, H. A fast modular simulator for combinational logic circuits generated by genetic algorithm. In: **Evolutionary Computation (CEC), 2013 IEEE Congress on**. [S.l.: s.n.], 2013. p. 2797–2801.

CANCARE, F.; SANTAMBROGIO, M.; SCIUTO, D. A direct bitstream manipulation approach for virtex4-based evolvable systems. In: **Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on**. [S.l.: s.n.], 2010. p. 853–856.

CHEANG, S. M.; LEE, K. H.; LEUNG, K. S. Applying genetic parallel programming to synthesize combinational logic circuits. **Evolutionary Computation, IEEE Transactions on**, v. 11, n. 4, p. 503 –520, aug. 2007. ISSN 1089-778X.

CLAUS, C. et al. A new framework to accelerate virtex-ii pro dynamic partial self-reconfiguration. In: **Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International**. [S.l.: s.n.], 2007. p. 1–7.

COELLO, C.; ALBA, E.; LUQUE, G. Comparing different serial and parallel heuristics to design combinational logic circuits. In: **Evolvable Hardware, 2003. Proceedings. NASA/DoD Conference on**. [S.l.: s.n.], 2003. p. 3 – 12.

COELLO, C. A. C.; AGUIRRE, A. H. Design of combinational logic circuits through an evolutionary multiobjective optimization approach. **Artif. Intell. Eng. Des. Anal. Manuf.**, Cambridge University Press, New York, NY, USA, v. 16, n. 1, p. 39–53, jan. 2002. ISSN 0890-0604. Disponível em: <http://dx.doi.org/10.1017/S0890060401020054>.

COELLO, C. C.; AGUIRRE, A.; BUCKLES, B. Evolutionary multiobjective design of combinational logic circuits. In: **Evolvable Hardware, 2000. Proceedings. The Second NASA/DoD Workshop on**. [S.l.: s.n.], 2000. p. 161–170.

DONTHI, S.; HAGGARD, R. A survey of dynamically reconfigurable fpga devices. In: **System Theory, 2003. Proceedings of the 35th Southeastern Symposium on**. [S.l.: s.n.], 2003. p. 422–426. ISSN 0094-2898.

FERREIRA, C. **Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence**. 2nd. ed. [S.l.]: Springer, 2006. ISBN 3-540-32796-7.

GAN, Z. et al. Evolutionary design of combinational logic circuits using an improved gene expression-based clonal selection algorithm. In: **Natural Computation, 2009. ICNC '09. Fifth International Conference on**. [S.l.: s.n.], 2009. v. 4, p. 37–41.

GUAN, Z. et al. Exploiting stochastic delay variability on fpgas with adaptive partial rerouting. In: **Field-Programmable Technology (FPT), 2013 International Conference on**. [S.l.: s.n.], 2013. p. 254–261.

HERNANDEZ-AGUIRRE, A.; BUCKLES, B.; COELLO-COELLO, C. Gate-level synthesis of boolean functions using binary multiplexers and genetic programming. In: **Evolutionary Computation, 2000. Proceedings of the 2000 Congress on**. [S.l.: s.n.], 2000. v. 1, p. 675 –682 vol.1.

HERRERA-ALZU, I.; LOPEZ-VALLEJO, M. System design framework and methodology for xilinx virtex fpga configuration scrubbers. **Nuclear Science, IEEE Transactions on**, v. 61, n. 1, p. 619–629, Feb 2014. ISSN 0018-9499.

HIGUCHI, T. et al. Evolvable hardware with genetic learning. In: **Circuits and Systems, 1996. ISCAS '96., Connecting the World., 1996 IEEE International Symposium on**. [S.l.: s.n.], 1996. v. 4, p. 29–32 vol.4.

HORI, Y. et al. Sasebo-giii: A hardware security evaluation board equipped with a 28-nm fpga. In: **Consumer Electronics (GCCE), 2012 IEEE 1st Global Conference on**. [S.l.: s.n.], 2012. p. 657–660.

ICHINOMIYA, Y. et al. Designing flexible reconfigurable regions to relocate partial bitstreams. In: **Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on**. [S.l.: s.n.], 2012. p. 241–241.

JACOBS, A. et al. Reconfigurable fault tolerance: A comprehensive framework for reliable and adaptive fpga-based space computing. **ACM Trans. Reconfigurable Technol. Syst.**, ACM, New York, NY, USA, v. 5, n. 4, p. 21:1–21:30, dez. 2012. ISSN 1936-7406. Disponível em: <http://doi.acm.org/10.1145/2392616.2392619>.

KARNAUGH, M. The map method for synthesis of combinational logic circuits. **American Institute of Electrical Engineers, Part I: Communication and Electronics, Transactions of the**, v. 72, n. 5, p. 593–599, Nov 1953. ISSN 0097-2452.

KE, P.; NIE, X.; CAO, Z. A generic architecture of complete intrinsic evolvable digital circuits. In: **Knowledge Acquisition and Modeling (KAM), 2010 3rd International Symposium on**. [S.l.: s.n.], 2010. p. 379–382.

KOCH, D.; BECKHOFF, C.; TøRRISON, J. Advanced partial run-time reconfiguration on spartan-6 fpgas. In: **Field-Programmable Technology (FPT), 2010 International Conference on**. [S.l.: s.n.], 2010. p. 361–364.

KOZA, J. R. **Genetic Programming: On the Programming of Computers by Means of Natural Selection**. Cambridge, MA, USA: MIT Press, 1992. ISBN 0-262-11170-5.

KWOK, T.-O.; KWOK, Y.-K. On the design of a self-reconfigurable sopc cryptographic engine. In: **Distributed Computing Systems Workshops, 2004. Proceedings. 24th International Conference on**. [S.l.: s.n.], 2004. p. 876–881.

LEVI, D.; GUCCIONE, S. Geneticfpga: evolving stable circuits on mainstream fpga devices. In: **Evolvable Hardware, 1999. Proceedings of the First NASA/DoD Workshop on**. [S.l.: s.n.], 1999. p. 12–17.

LOHN, J. et al. Evolutionary design of an x-band antenna for nasa's space technology 5 mission. In: **Antennas and Propagation Society International Symposium, 2004. IEEE**. [S.l.: s.n.], 2004. v. 3, p. 2313–2316 Vol.3.

LOPEZ, B. et al. Power-aware multi-objective evolvable hardware system on an fpga. In: **Adaptive Hardware and Systems (AHS), 2014 NASA/ESA Conference on**. [S.l.: s.n.], 2014. p. 61–68.

MALIK, S. Analysis of cyclic combinational circuits. In: **Computer-Aided Design, 1993. ICCAD-93. Digest of Technical Papers., 1993 IEEE/ACM International Conference on**. [S.l.: s.n.], 1993. p. 618–625.

MCCLUSKEY, E. J. Minimization of Boolean functions. **The Bell System Technical Journal**, v. 35, n. 5, p. 1417–1444, nov. 1956.

MILLER, J. F. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In: BANZHAF, W. et al. (Ed.). **Proceedings of the Genetic and Evolutionary Computation Conference**. Orlando, Florida, USA: Morgan Kaufmann, 1999. v. 2, p. 1135–1142. ISBN 1-55860-611-4.

MILLER, J. F.; JOB, D.; VASSILEV, V. K. Principles in the evolutionary design of digital circuits-part i. **Genetic Programming and Evolvable Machines**, Kluwer Academic Publishers, Hingham, MA, USA, v. 1, n. 1-2, p. 7–35, abr. 2000. ISSN 1389-2576. Disponível em: <http://dx.doi.org/10.1023/A:1010016313373>.

MUKHERJEE, B.; DANDAPAT, A. Design of combinational circuits by cyclic combinational method for low power vlsi. In: **Electronic System Design (ISED), 2010 International Symposium on**. [S.l.: s.n.], 2010. p. 107–112.

OREIFEJ, R. et al. Layered approach to intrinsic evolvable hardware using direct bitstream manipulation of virtex ii pro devices. In: **Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on**. [S.l.: s.n.], 2007. p. 299–304.

PATTERSON, C.; GUCCIONE, S. Jbits 8482; design abstractions. In: **Field-Programmable Custom Computing Machines, 2001. FCCM '01. The 9th Annual IEEE Symposium on**. [S.l.: s.n.], 2001. p. 251–252.

QUINE, W. V. A way to simplify truth functions. **The American Mathematical Monthly**, Mathematical Association of America, v. 62, n. 9, p. pp. 627–631, 1955. ISSN 00029890. Disponível em: <http://www.jstor.org/stable/2307285>.

REN, A. et al. Implement of evolable hardware based improved genetic algorithm. In: **Natural Computation (ICNC), 2011 Seventh International Conference on**. [S.l.: s.n.], 2011. v. 4, p. 2112–2115. ISSN 2157-9555.

RIEDEL, M.; BRUCK, J. The synthesis of cyclic combinational circuits. In: **Design Automation Conference, 2003. Proceedings**. [S.l.: s.n.], 2003. p. 163–168.

SALVADOR, R. et al. Evolvable 2d computing matrix model for intrinsic evolution in commercial fpgas with native reconfiguration support. In: **Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on**. [S.l.: s.n.], 2011. p. 184–191.

SALVADOR, R. et al. Implementation techniques for evolvable hw systems: virtual vs. dynamic reconfiguration. In: **Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on**. [S.l.: s.n.], 2012. p. 547–550.

SEKANINA, L. Virtual reconfigurable circuits for real-world applications of evolvable hardware. In: **Proceedings of the 5th International Conference on Evolvable Systems: From Biology to Hardware**. Berlin, Heidelberg: Springer-Verlag, 2003. (ICES'03), p. 186–197. ISBN 3-540-00730-X. Disponível em: <http://dl.acm.org/citation.cfm?id=1766731.1766753>.

SEKANINA, L. Evolutionary functional recovery in virtual reconfigurable circuits. **J. Emerg. Technol. Comput. Syst.**, ACM, New York, NY, USA, v. 3, n. 2, jul. 2007. ISSN 1550-4832. Disponível em: <http://doi.acm.org/10.1145/1265949.1265954>.

SEKANINA, L.; FRIEDL, S. An evolvable combinational unit for fpgas. **Computers and Artificial Intelligence**, v. 23, n. 5, p. 461–486, 2004. Disponível em: <http://dblp.uni-trier.de/db/journals/cai/cai23.html#SekaninaF04>.

SLOWIK, A.; BIALKO, M. Design and optimization of combinational digital circuits using modified evolutionary algorithm. In: **Artificial Intelligence and Soft Computing - ICAISC 2004, 7th International Conference**. [s.n.], 2004. v. 3070, p. 468–473. ISBN 3-540-22123-9. Http://www.odysci.com/article/1010112985275770. Disponível em: <http://link.springer.de/link/service/series/0558/bibs/3070/30700468.htm>.

SLOWIK, A.; BIALKO, M. Evolutionary design and optimization of combinational digital circuits with respect to transistor count. **Bulletin of the Polish Academy of Sciences**, v. 54, n. 4, p. 437–442, dez. 2006. ISSN 0239-7528. Disponível em: <http://bulletin.pan.pl/(54-4)437.html>.

SLOWIK, A.; BIALKO, M. Evolutionary design of combinational digital circuits: State of the art, main problems, and future trends. In: **Information Technology, 2008. IT 2008. 1st International Conference on**. [S.l.: s.n.], 2008. p. 1 –6.

THOMPSON, A. Silicon evolution. In: **Stanford University**. [S.l.]: MIT Press, 1996. p. 444–452.

VASICEK, Z.; SEKANINA, L. On area minimization of complex combinational circuits using cartesian genetic programming. In: **Evolutionary Computation (CEC), 2012 IEEE Congress on**. [S.l.: s.n.], 2012. p. 1–8.

VASSILEV, V.; MULLER, J.; FOGARTY, T. Digital circuit evolution and fitness landscapes. In: **Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on**. [S.l.: s.n.], 1999. v. 2, p. 3 vol. (xxxvii+2348).

VIJAYAKUMARI, C. et al. An improved design of combinational digital circuits with multiplexers using genetic algorithm. In: **Emerging Research Areas and 2013 International Conference on Microelectronics, Communications and Renewable Energy (AICERA/ICMiCR), 2013 Annual International Conference on**. [S.l.: s.n.], 2013. p. 1–5.

VIJAYAKUMARI, C.; MYTHILI, P. A faster 2d technique for the design of combinational digital circuits using genetic algorithm. In: **Power, Signals, Controls and Computation (EPSCICON), 2012 International Conference on**. [S.l.: s.n.], 2012. p. 1–5.

WANG, J. R.; WANG, D.; LAI, J. M. A hierarchical parallel evolvable hardware based on network on chip. In: **Reconfigurable Computing and FPGAs (ReConFig), 2013 International Conference on**. [S.l.: s.n.], 2013. p. 1–6.

XILINX. **Using a Microprocessor to Configure Xilinx FPGAs via Slave Serial or SelectMAP Mode. XAPP502, (v1.6.1) edition**. August 2009. Disponível em: <http://www.xilinx.com/support/documentation/application_notes/xapp502.pdf>.

XILINX. **Spartan-6 FPGA Configurable Logic Block User Guide. UG384, (v1.1) edition**. February 2010. Disponível em: <http://www.xilinx.com/support/documentation/user_guides/ug384.pdf>.

XILINX. **7 Series FPGAs Overview. Product Specification. DS180, (v1.16) edition**. October 2014. Disponível em: <http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf>.

XILINX. **Zynq-7000 All Programmable SoC Overview. Preliminary Product Specification. DS190, (v1.7) edition**. October 2014. Disponível em: <http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf>.

YU, H.; XU, Q.; LEONG, P. On timing yield improvement for fpga designs using architectural symmetry. In: **Field Programmable Logic and Applications (FPL), 2011 International Conference on**. [S.l.: s.n.], 2011. p. 539–544.