MARCO CAETANO LAZARINE BONTORIN

UMA TÉCNICA BIO-INSPIRADA DE POSICIONAMENTO, SEGMENTAÇÃO E RETEMPORIZAÇÃO PARA REDUZIR O CONSUMO DE ENERGIA EM FPGAS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Prof. Aldri Luiz dos Santos
Coorientadora: Profa. Michele Nogueira Lima

CURITIBA

2011

MARCO CAETANO LAZARINE BONTORIN

UMA TÉCNICA BIO-INSPIRADA DE POSICIONAMENTO, SEGMENTAÇÃO E RETEMPORIZAÇÃO PARA REDUZIR O CONSUMO DE ENERGIA EM FPGAS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Prof. Aldri Luiz dos Santos
Coorientadora: Profa. Michele Nogueira Lima

CURITIBA

2011

Bontorin, Marco Caetano Lazarine
Uma técnica bio-inspirada de posicionamento, segmentação e retemporização para reduzir o consumo de energia em FPGAS / Marco Caetano Lazarine Bontorin. – Curitiba, 2011.
66 f.: il., tab.

Dissertação (Mestrado) - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática.
Orientador: Aldri Luiz dos Santos
Coorientadora: Michele Nogueira Lima

1. Circuitos integrados digitais. 2.Algoritmos. 3. Energia – Consumo. I. Santos, Aldri Luiz dos. II. Lima, Michele Nogueira. III. Universidade Federal do Paraná. IV. Título.

CDD: 621.395

DEDICATÓRIA

Dedico este trabalho em especial a meus pais, Maria e Agostinho, e também a meu irmão, Marcelino e minhas irmãs Célia, Maria Salete, Mauria e Marizângela.

AGRADECIMENTOS

Em primeiro lugar agradeço a Deus por ter me dado forças nos momentos em que achei que não conseguiria ir em frente. Agradeço ao apoio dos meus familiares, em especial minha mãe Maria e meu pai Agostinho, que sempre nos momentos difíceis me deram suporte e tranquilidade para que eu pudesse fazer o meu trabalho. Agradeço ao governo federal que, através da CAPES e da Universidade Federal do Paraná, financiou as minhas atividades de pesquisa, possibilitando minha dedicação exclusiva ao mestrado.

Agradeço a meus orientadores e amigos, Aldri e Michele, por todo o tempo de orientação e embasamento em pesquisa que foram de extrema importância para a realização deste trabalho. Além disso, meus orientadores tiveram papel decisivo no direcionamento inicial da pesquisa, período que foi o mais difícil pra mim por uma série de problemas emocionais. Por isso, a ajuda deles além de proporcionar a minha boa formação, também garantiram minha sanidade mental ao final do mestrado.

Agradeço muito aos membros do grupo NR2, onde tive um excelente ambiente de estudo e trabalho e fiz muitas amizades. O grupo NR2 ofereceu ótima infraestrutura de pesquisa através do seu ambiente de grupo e o seu laboratório de pesquisa. Neste ambiente, houve muito trabalho, mas também muita descontração, risadas, nerdices e os conselhos fundamentais da Celite nos momentos de aperto. São muitas pessoas para agradecer, se esqueci de alguém peço desculpas, obrigado por tudo!

RESUMO

Os *Field-Programmable Gate Arrays (FPGAs)* podem beneficiar aplicações que apresentam requisitos de tempo real, pois estas podem executar de forma mais rápida em *hardware*. Além da velocidade, os FPGAs fornecem flexibilidade, facilitando a implementação novas funcionalidades. No entanto, dispositivos com restrições de recursos, como nós sensores de redes corporais, além de atender requisitos de tempo real precisam economizar energia. Para que tais dispositivos possam se beneficiar das vantagens dos FPGAs, são necessárias técnicas que otimizem o seu consumo de energia. As técnicas existentes de redução do consumo de energia envolvem desde novas tecnologias de transístores a algoritmos de mapeamento de circuitos. Esses algoritmos são usados para otimizar projetos em linguagens de descrição de *hardware* nas diferentes etapas do fluxo CAD para que o circuito final consuma menos energia. Contudo, essas técnicas em geral não conseguem integrar uma otimização em várias etapas do fluxo CAD de FPGAs.

Neste trabalho é proposta uma técnica bio-inspirada, chamada de **AntES** (*pl**A**cement, pipeli**N**ing and re**T**iming for **E**nergy **S**aving*). Esta técnica atua no fluxo CAD de FPGAs, agregando posicionamento, segmentação e retemporização para a redução do consumo de energia do circuito final. A **AntES** tem como base a especificação de um modelo bio-inspirado, consistindo de equações que definem o comportamento de cada agente e os algoritmos que descrevem os detalhes das suas operações. A técnica consiste de agentes de posicionamento, que buscam aproximar os blocos cujas interconexões são mais ativas, e agentes de retemporização, que visam balancear os atrasos entre os registradores. Como o *glitching* é uma importante fonte de potência dissipada desnecessária, segmentação e retemporização contribuem para a redução do consumo de energia.

Uma avaliação de desempenho compara em dois cenários o posicionamento da técnica **AntES** ao do VPR original considerando a potência dissipada, o atraso do caminho crítico e o tamanho da caixa delimitadora. O primeiro cenário utiliza um conjunto de circuitos *benchmarks* genéricos, independente de sua aplicabilidade. O segundo cenário emprega um conjunto de *benchmarks* cuja funcionalidade está presente em nós sensores de redes corporais. Os resultados são similares em ambos os cenários e apresentam reduções no consumo de energia para a maioria dos circuitos *benchmarks* utilizados, salvo para alguns circuitos sequenciais de grandes áreas. Este problema pode ser corrigido com a implementação da segmentação, pois a mesma reduz o *glitching* em circuitos sequenciais. No entanto, como a retemporização da técnica **AntES** não pode ser implementada, seus efeitos sobre os resultados do posicionamento ainda não puderam ser avaliados.

Palavras-chave: FPGA, economia de energia, técnicas bio-inspiradas.

ABSTRACT

Field-Programmable Gate Arrays (FPGAs) can benefit applications that demand real-time requirements, because these applications run faster when implemented in *hardware*. In addition to speed, FPGAs provide flexibility as they can be reconfigured, facilitating the implementation of new features. However, resource-constrained devices, like sensor nodes in body area networks, need to meet both real-time processing and energy saving requirements. In order to take advantage of the FPGA features, techniques are needed to optimize the FPGA's energy consumption. Existing techniques to reduce energy consumption are based on several levels of abstraction, from new transistor technologies to circuit mapping algorithms. These algorithms are used to optimize designs in hardware description languages at different stages of the FPGA CAD flow. However, such techniques often fail to integrate a multi-step optimization of the FPGA CAD flow.

This work proposes a bio-inspired technique, called **AntES** (*pl**A**cement, pipeli**N**ing and re**T**iming for **E**nergy **S**aving*). This technique focuses on the FPGA CAD Flow, joining placement, pipelining and retiming in order to reduce FPGA's energy consumption of a given circuit. **AntES** is based on a bio-inspired model specification, providing equations that define the behaviour of bio-inspired agents and their operations. It provides placement agents and retiming agents. The placement agents focus on reducing the distance between blocks in which there are interconections with high activity. On the other hand, the retiming agents must balance the interconection delay between the circuit registers. The goal of this delay balancing is reducing the clock period, but the resulting signal stability reduces the circuit glitching as well. As the glitching is an important source of unnecessary power dissipation, pipelining and retiming contribute in energy consumption reduction.

The performance evaluation compares the **AntES** placement performance with the original VPR placement performance considering power dissipation, critical path delay and bounding box size. It considers two cenarios, the first one uses an MCNC benchmarks subset, without considering any specific application. The second cenario takes into account a specific MCNC benchmark subset, whose the basic operations are performed in body area networks. The results are similar in both cenarios and show energy consumption reduction for the most benchmarks, except for the largest sequential circuits. This problem can be solved with pipelining implementation, because it reduces glitching in sequential circuits. However, as the **AntES** retiming could not be implemented, so its impact over the results could not be evaluated yet.

Keywords: FPGA, energy saving, bio-inspired techniques.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ACE	ACtivity Estimation
AntES	plAcement, pipeliNing and reTiming for Energy Saving
ASIC	Application-Specific Integrated Circuit
BAN	Body Area Network
BLE	Basic Logic Element
CAD	Computer-aided Design
CMOS	Complementary Metal-Oxide-Semiconductor
EDIF	Electronic Design Interchange Format
FPGA	Field-Programmable Gate Array
LUT	Look-Up Table
MCNC	Microelectronics Center of North Carolina
SA	Simulated Annealing
SRAM	Static Random Access Memory
T-VPack	Timing-driven Versatile Packing
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
VPR	Versatile Place and Route

NOTAÇÃO

A_{iob}	agente de posicionamento dos blocos de entrada e saída
A_{lb}	agente de posicionamento dos blocos lógicos
A_{ret}	agente de retemporização
f_{iob}	frequência de movimentação do agente A_{iob}
f_{lb}	frequência de movimentação do agente A_{lb}
f_{ret}	frequência de movimentação do agente A_{ret}
$pPegar$	probabilidade de pegar
$pDeixar$	probabilidade de deixar
P	potência dissipada
T	atraso das conexões
W	comprimento das conexões
P_{gen}	Probabilidade de gerar um agente
ag	um agente
$agCp$	cópia de um agente
T_{loc}	atraso do caminho local
T_{gl}	menor atraso registrado entre dois registradores
T_{ant}	atraso do caminho local anterior
ble_{at}	bloco básico atual
bl_{or}	bloco lógico de origem
bl_{d}	bloco lógico de destino
bl_{at}	bloco lógico atual
ble_{d}	bloco básico de destino

CAPÍTULO 1

INTRODUÇÃO

A utilização de dispositivos computacionais portáteis vem crescendo a cada dia. Esses dispositivos geralmente utilizam tecnologias de comunicação sem fio auxiliadas por protocolos e infraestrutura de comunicação já existentes. A comunicação em uma rede sem fio pode ser feita através de um ponto de acesso ou através de conexão direta entre os dispositivos (nós). As redes cuja comunicação entre os nós é direta sem uma infraestrutura de suporte são chamadas de redes *ad-hoc*. As redes de sensores sem fio são um exemplo de redes *ad-hoc*. Essas redes são compostas por vários nós aplicados em atividades específicas que exigem monitoramento constante como o acompanhamento de fatores climáticos, do tráfego de veículos e do comportamento de algumas espécies animais.

Um **nó sensor** contém basicamente um transmissor/receptor e um ou mais sensores. O transmissor/receptor serve para a comunicação entre os nós através de ondas de rádio de forma que os sensores possam enviar os dados coletados até uma estação base. Os dispositivos sensores são responsáveis por coletar informações do meio, como umidade, temperatura, pressão atmosférica e outras. As redes de sensores podem ser aplicadas em diferentes ambientes, e muitas vezes não é possível ter acesso físico aos nós para que sua fonte de energia seja substituída manualmente. A falta de energia faz com que um nó sensor deixe de operar e quando uma certa quantidade de nós fica fora de operação, a relevância dos dados coletados pode ser comprometida.

Em geral, os nós sensores são equipados com uma unidade de processamento e executam um sistema operacional simples. Esses sistemas operacionais são responsáveis por gerenciar os recursos do nó sensor, como periféricos, memória e interface de rádio. Dessa forma, os nós sensores podem suportar determinados protocolos que levam em conta o consumo de energia, e algoritmos de agregação de dados. Estes algoritmos de agregação de dados são usados para reduzir o número de retransmissões, e assim economizar energia.

Um subtipo de rede de sensores é a rede corporal sem fio. Esta monitora os sinais do corpo humano, como a atividade elétrica muscular, a dilatação dos vasos e a movimentação do corpo [1]. Logo, se alguma alteração significativa nos padrões desses sinais indicar uma doença, a rede poderá ser configurada para responder adequadamente a seus sintomas, auxiliando na sua prevenção.

Para que as redes corporais sem fio possam ser largamente adotadas elas devem atender requisitos fundamentais, como a relevância dos sinais capturados, a segurança na comunicação, a privacidade e a facilidade de uso. Atender esses requisitos demanda processamento de sinais, criptografia, implementação de protocolos de comunicação, dentre

outras tarefas. Além disso, o processamento dessas tarefas deve fornecer resposta em tempo real [2].

Os nós sensores das redes corporais geralmente usam microcontroladores e periféricos de dimensão reduzida, pois precisam ser rápidos, flexíveis e eficientes em termos de consumo de energia. Os microcontroladores, embora ofereçam uma grande flexibilidade, têm limitações na velocidade de processamento quando precisam considerar várias tarefas. Uma alternativa para aumentar a velocidade de processamento é a implementação dedicada em *hardware* através de um circuito integrado de aplicação específica (*Application-Specific Integrated Circuit - ASIC*) [3]. Os ASICs costumam atender bem as necessidades de uma aplicação em termos de área, velocidade e consumo de energia, pois podem ser projetados e otimizados para isso. O problema de tais circuitos é a falta de flexibilidade, visto que se os requisitos da aplicação mudarem, um outro circuito deverá ser fabricado para atendê-los.

Uma solução intermediária para atender os requisitos das redes corporais é a utilização de circuitos programáveis nos nós sensores. Os circuitos programáveis agrupam componentes que podem implementar diferentes funções lógicas. Além disso, eles permitem que o usuário final defina a função a ser implementada no circuito. Dessa forma, os circuitos programáveis são mais rápidos que os microcontroladores e mais flexíveis que os ASICs.

Dentre os circuitos programáveis existentes hoje, os FPGAs são largamente utilizados por permitirem a reprogramação de sua funcionalidade, alguns deles durante a execução da aplicação [4]. As características dos FPGAs oferecem meios para atender os requisitos das redes corporais, pois diferentes aplicações podem ser executadas concorrentemente entre si [5, 6, 7]. Cada aplicação implementada tem a vantagem individual do paralelismo de *hardware* na sua execução. Além disso, pode-se seletivamente modificar a funcionalidade das aplicações em tempo de execução, adaptando o sistema a eventuais mudanças. Este tipo de implementação tira vantagem do paralelismo inerente ao *hardware* e pode implementar os requisitos das redes corporais de forma mais rápida.

Entretanto, existe um obstáculo para o uso dos FPGAs em redes corporais [8, 9, 10]. Para que uma rede corporal possa monitorar o paciente corretamente, este precisa estar em contato com a rede durante todo o período necessário para a captura dos sinais relevantes na determinação do seu estado. Falhas nos nós sensores que os deixem inoperantes ou operando de forma inadequada podem resultar na perda ou no corrompimento de dados, colocando o paciente em risco, por exemplo. A falta de energia é uma das falhas que tornam os nós sensores inoperantes. Contudo, a redução do consumo de energia dos dispositivos para prolongar a vida útil da rede tem sido um desafio nas redes corporais [2]. Os FPGAs, apesar de serem rápidos e flexíveis, consomem mais energia que os microcontroladores e os ASICs devido ao seus componentes reconfiguráveis [3].

Portanto, a larga utilização de FPGAs em dispositivos portáteis como os nós sensores depende de técnicas que reduzam o seu consumo de energia. Atualmente, várias pesqui-

sas têm sido desenvolvidas no meio acadêmico e empresarial, focando em técnicas para economia de energia em FPGAs [11]. Essas técnicas envolvem desde novas tecnologias de transístores a projetos em linguagens de descrição de *hardware* otimizados para consumir menos energia. Este trabalho propõe uma técnica para economizar energia em FPGAs desenvolvida para integrar o fluxo de projeto de uma ferramenta CAD.

1.1 Problema

O problema tratado neste trabalho consiste em reduzir o consumo de energia em FPGAs, em especial o consumo gerado pelo roteamento e pelas transições indesejadas nos sinais. O consumo de energia nos FPGAs pode ser dividido em consumo de energia estático e consumo de energia dinâmico. Este trabalho tem como foco o consumo de energia dinâmica por representar aproximadamente 60% do consumo de energia total [11]. A energia dinâmica compreende a energia dissipada no circuito de roteamento, a energia dissipada na rede de *clocks* e a energia dissipada pelos blocos lógicos [12]. Dentre esses elementos, o que mais consome energia é o circuito de roteamento.

O circuito de roteamento de propósito geral nos FPGAs é composto de trilhas horizontais e verticais. Cada trilha é composta de vários segmentos que são conectados entre si utilizando caixas comutadoras. Por sua vez, os blocos lógicos se conectam às trilhas de roteamento através de caixas de conexão. A energia gasta no roteamento está relacionada diretamente à **distância entre os blocos lógicos**, isto é, por quantos segmentos e conexões programáveis o sinal deve passar até atingir o seu destino.

A energia dinâmica em FPGAs é gasta devido a carga e descarga de capacitores durante a execução do circuito [12]. A quantidade de energia gasta, portanto, é proporcional a taxa de transição dos sinais no circuito. Para identificar quais componentes e interconexões têm maior taxa de transição são usadas ferramentas de estimativa de atividade. Muitas vezes, no entanto, ocorrem **transições indesejadas** de um sinal de saída devido ao **atraso desigual de propagação** das entradas de um componente do FPGA. Essa transição não prevista, chamada de *glitching*, é responsável por 31% do consumo de energia dinâmica do FPGA [13].

1.2 Objetivos

Este trabalho propõe uma técnica bio-inspirada de posicionamento, segmentação (*pipelining*) e retemporização (*retiming*) para reduzir o consumo de energia em FPGAs. Essa técnica tem como base o comportamento de colônias de formigas para posicionar o circuito, criar e reposicionar registradores. O posicionamento do circuito e o reposicionamento de registradores têm como entradas as coordenadas que refletem a arquitetura do FPGA e os elementos, blocos lógicos e seus registradores internos, que serão

posicionados. As coordenadas de cada elemento são modificadas com base em informações locais relacionadas a qualidade das conexões em termos de consumo de energia.

Nessa técnica o cálculo do posicionamento para um elemento é independente do cálculo para o posicionamento dos demais, permitindo que o posicionamento e a retemporização possam ser feitos simultaneamente. Além disso, a frequência relativa desses cálculos, considerando os diferentes tipos de componentes, pode ser ajustada para que um posicionamento satisfatório seja alcançado. A idéia parte do princípio de que a partir de tarefas simples e informações locais, uma colônia de formigas trabalhando em conjunto resolve problemas mais complexos. Um exemplo clássico inspirado em colônia de formigas é a solução aproximada para o problema de encontrar o melhor caminho entre dois vértices em um grafo [14]. De forma análoga, este trabalho pretende alcançar seu objetivo se inspirando no comportamento de colônias de formigas, através de agentes (trecho de código em *software*), realizando cálculos individuais simples.

Fundamentado nesta abordagem, este trabalho apresenta uma técnica agregada de posicionamento, segmentação e retemporização chamada **AntES** (*pl**A**cement, pipeli**N**ing and re**T**iming for **E**nergy **S**aving*). Seu objetivo é reduzir o consumo de energia em FPGAs e possibilitar o uso dessa tecnologia em redes corporais. A etapa de posicionamento pretende economizar energia reduzindo o atraso das conexões mais ativas através da aproximação dos blocos lógicos de origem e destino de uma conexão. A segmentação e a retemporização pretendem reduzir as transições indesejadas dos sinais do FPGA, diminuindo o atraso entre os registradores no mesmo caminho. Como as transições de sinal estão diretamente relacionadas com a potência dinâmica dissipada, reduzir o número de transições diminui o consumo de energia. Pretende-se mostrar que as duas técnicas combinadas, executando simultaneamente, resultam em uma economia de energia significativa.

1.3 Contribuições

O desenvolvimento e a avaliação da técnica bio-inspirada de posicionamento, segmentação e retemporização **AntES** contém as seguintes contribuições:

- Um estado da arte dos FPGAs e a redução do seu consumo de energia, cujos resultados aproximam os FPGAs do uso em aplicações com restrições de energia.
- Especificação do algoritmo de posicionamento de circuitos em FPGAs da técnica **AntES**. Neste algoritmo, agentes de posicionamento trabalham em conjunto para aproximar os blocos lógicos cujas interconexões têm uma atividade dinâmica maior.
- Especificação do algoritmo de retemporização de circuitos em FPGAs da técnica **AntES**. Os agentes de retemporização utilizam este algoritmo para balancear os atrasos entre os registradores de um dado circuito.

- Implementação do posicionamento **AntES** na ferramenta de posicionamento e roteamento VPR, substituindo o algoritmo de posicionamento nativo pelo algoritmo de posicionamento da técnica **AntES**.

- Avaliação do algoritmo de posicionamento, comparando os seu resultados ao obtido pelo VPR em um contexto genérico, bem como em um cenário fictício de redes corporais.

1.4 Estrutura da dissertação

Esta dissertação está organizada em cinco capítulos. O Capítulo 2 apresenta os fundamentos de FPGAs, as técnicas empregadas para efetuar o posicionamento, a retemporização e as técnicas usadas para economizar energia. Uma taxonomia das técnicas de economia de energia em FPGAs é apresentada, organizando os trabalhos relacionados existentes na literatura. O Capítulo 3 descreve a técnica **AntES** que visa economizar energia em FPGAs utilizando uma abordagem bio-inspirada. O Capítulo 4 mostra os resultados obtidos pela técnica **AntES** comparados aos obtidos pelo VPR. Por fim, o Capítulo 5 apresenta as conclusões e os trabalhos futuros.

CAPÍTULO 2

FUNDAMENTOS

Este capítulo apresenta uma discussão sobre os conceitos relacionados aos FPGAs, sua arquitetura e seus principais componentes. Na Seção 2.1 são discutidos detalhes da tecnologia e arquitetura dos FPGAs. Ela ilustra as diferentes etapas do fluxo de projeto de um FPGA, com destaque para etapa de posicionamento, e define a retemporização (*retiming*) de circuitos em FPGAs. Além disso, ela descreve o modelo de energia, utilizado neste trabalho para realizar as estimativas de consumo de energia de diferentes alternativas de arquiteturas implementadas nos FPGAs. As principais técnicas de otimização do consumo de energia em FPGAs são apresentadas na Seção 2.2. Por fim, a Seção 2.3 apresenta os trabalhos correlatos que utilizam técnicas de inteligência artificial, em especial o campo da inteligência de enxames, que dá suporte à solução proposta neste trabalho.

2.1 *Field-Programmable Gate Arrays (FPGAs)*

Os FPGAs são constituídos de um arranjo de blocos lógicos reconfiguráveis interconectados entre si por ligações diretas dedicadas ou por linhas de roteamento de propósito geral. Sua funcionalidade pode ser definida pelo usuário, modelando o circuito em uma linguagem de descrição de *hardware* e posterior programação dos blocos lógicos através de uma ferramenta CAD. A programação do FPGA é realizada através de *bits* de configuração utilizados para preencher as tabelas de consulta (*Look-Up Tables - LUTs*), definir linhas de saída dos multiplexadores e *gates* de transístores programáveis. As três tecnologias mais utilizadas para possibilitar a (re)programação dos circuitos em um FPGA são: *antifuse*, *flash* e SRAM.

O *antifuse* é basicamente um material isolante entre dois fios que uma vez submetido a uma alta voltagem passa a conduzir corrente. O *antifuse* apresenta a vantagem de ser não-volátil, ter baixa resistência e seu circuito não ocupar área do FPGA. Contudo, grandes transístores são necessários para fornecer a corrente necessária para programá-lo, o que reduz sua eficiência apesar de ainda ser a tecnologia que ocupa a menor área [15]. A célula de memória *flash* consiste basicamente de um *floating gate transistor*. Ela é não-volátil e ao contrário do *antifuse*, é reprogramável, tendo uma área superior a do *antifiuse*, mas bem menor que a ocupada pela SRAM. A célula de memória SRAM consiste de 6 transístores. Apesar de possuir a maior área e ser volátil, a SRAM é construída utilizando a tecnologia CMOS e por isso, beneficia-se da constante evolução dessa tecnologia que é padrão no mercado. Ao contrário das tecnologias discutidas anteriormente, a SRAM pode ser reprogramada um número ilimitado de vezes. Essas duas vantagens fizeram da

SRAM a tecnologia de programação de FPGAs dominante no mercado.

A Figura 2.1 ilustra uma estrutura comum de FPGAs de forma resumida. Nesta figura pode-se distinguir os blocos de I/O usados para efetuar as entradas e saídas dos FPGAs e os blocos lógicos que implementam funções definidas pelo usuário. As caixas de conexão são usadas para efetuar a conexão dos blocos lógicos e blocos de I/O aos segmentos do roteamento de propósito geral. As caixas comutadoras, por sua vez, conectam os segmentos entre si. Tanto as conexões entre os segmentos quanto as conexões dos segmentos aos blocos são programáveis. Atualmente, os FPGAs são mais complexos do que o representado na Figura 2.1. Eles contêm também vários circuitos de aplicação específica como, por exemplo, os blocos de memória RAM e os multiplicadores. Os circuitos específicos são mais eficientes do que os equivalentes implementados usando blocos lógicos reconfiguráveis.

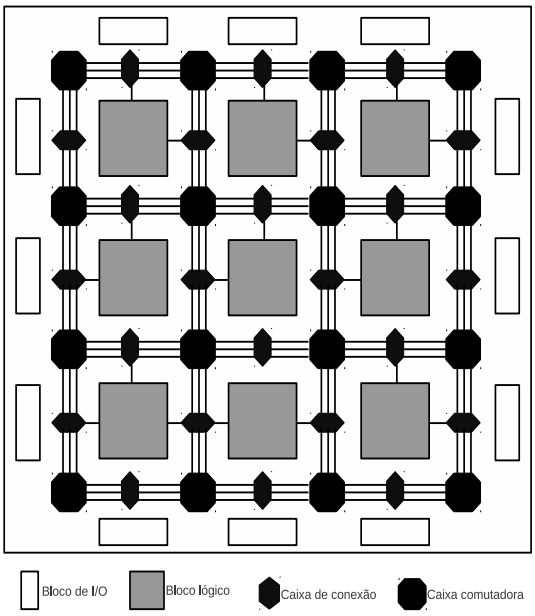


Figura 2.1: Estrutura de um FPGA

Os blocos lógicos mais simples são compostos basicamente de uma LUT, um *flip-flop* e um multiplexador. Uma LUT de quatro entradas, por exemplo, pode implementar qualquer função booleana de quatro variáveis e são elementos combinacionais programáveis muito utilizados nos FPGAs de hoje. O multiplexador permite selecionar entre a saída combinacional da LUT ou a saída do registrador que contém a informação da saída da LUT gravada no último ciclo de *clock*. Muitos trabalhos foram desenvolvidos visando agregar mais funcionalidade aos blocos lógicos de forma a evitar o uso do roteamento de propósito geral [16, 17, 11].

Em um desses estudos, Ahmed [16] demonstra que o aumento do tamanho da LUT reduz a área de um bloco lógico, pois menos LUTs implementam a mesma funcionalidade. Essa redução se dá até certo ponto (5 entradas), a partir do qual a área começa a aumentar,

pois o aumento do número de entradas da LUT gera necessidade de criar novas células SRAM e multiplexadores, aumentando sua área exponencialmente. O aumento do número de entradas na LUT também gera a necessidade de aumentar as linhas de roteamento que se ligam às entradas e saídas do bloco lógico, demonstrando que LUTs de 4 ou 5 entradas são as mais eficientes em área. Da mesma forma, uma LUT maior agrega mais funcionalidades, reduzindo o número de blocos lógicos no caminho crítico. Por outro lado, os atrasos internos podem ser tão grandes que não compensem a redução no atraso nas linhas de roteamento de propósito geral. Ahmed demonstra que LUTs com número de entradas entre 4 e 6 fornecem os melhores resultados em termos de área e velocidade [16].

Outra forma de se aumentar a granularidade de um bloco lógico é o agrupamento de vários blocos básicos (LUT, *flip-flop*, multiplexador) em um único bloco lógico. As interconexões entre os elementos dentro de um bloco lógico são mais rápidas, sendo uma forma de evitar o roteamento de propósito geral. Assim como o aumento no número de entradas por LUT, o aumento do número de blocos básicos por bloco lógico reduz o número de blocos lógicos no caminho crítico. Os estudos feitos por Ahmed também apontam que *clusters* de 3 a 10 blocos básicos como tendo os melhores resultados em termos de área e velocidade.

Assim que vários blocos básicos são agrupados em um bloco lógico, surge a necessidade de aumentar o número de entradas para um bloco lógico. Se por um extremo uma entrada do bloco lógico para cada entrada dos blocos básicos internos iria consumir muitos recursos de roteamento, poucas entradas no bloco lógico fazem com que alguns blocos básicos sejam subutilizados. Betz e Rose [17] estudaram detalhadamente esse problema e mostram a quantidade de blocos básicos que devem ser agrupados em um *cluster* lógico e o número de entradas para esse *cluster* a fim de obter melhor eficiência em área mantendo uma boa utilização dos blocos básicos. Seus resultados mostraram que para obter uma utilização de 98% da lógica dos blocos básicos, *clusters* de 4 a 8 blocos básicos mostraram-se mais eficientes em área que os com 2 blocos básicos.

Além das formas mostradas anteriormente, uma outra alternativa é a implementação de conexões diretas entre blocos lógicos vizinhos [18]. Essas conexões amenizam o problema, pois conectam blocos vizinhos (vizinhança 4 ou 8) por fios dedicados de velocidade mais alta. Além dessa alternativa, algumas otimizações foram propostas para tornar o roteamento mais eficiente quando as linhas de roteamento de propósito geral não puderem ser evitadas. As caixas comutadoras permitem a implementação de muitos padrões de interconexão. Essa flexibilidade tem o preço de um *hardware* extra que adiciona atraso no caminho crítico. Uma mistura dos elementos reconfiguráveis tradicionais e caminhos fixos de metal nas caixas comutadoras foi proposta em [19], tornando essas conexões mais rápidas. Nesse trabalho os autores verificaram o perfil de conexões utilizadas nas caixas comutadoras para que os padrões mais frequentes pudessem ser implementados como caminhos fixos (fios de metal fixos e não programáveis).

Pelos atrasos variáveis na comunicação entre os blocos lógicos através do roteamento de propósito geral, os FPGAs costumam operar em frequências de *clock* bem abaixo da frequência de microprocessadores. Em [20], Singh e Brown propõem a adição de registradores nas caixas comutadoras para aumentar a frequência do *clock*. Eles analisaram também os compromissos entre o desempenho obtido e a área adicional, mostrando como resultado um aumento de 25% na velocidade com 10% de área adicional.

Alguns FPGAs atuais, como por exemplo o Virtex-II Pro [21], permitem a reconfiguração de partes de seu circuito durante sua execução de forma que é possível a circuitos não afetados continuarem sua execução normalmente. Essas características tornam o FPGA uma plataforma ideal para a prototipagem de novos circuitos e testes de diferentes arquiteturas antes delas serem efetivamente implementadas. Atualmente, há também uma crescente utilização dos FPGAs como dispositivos finais. Um resumo de aplicações implementadas em FPGAs é encontrado em [4]. A capacidade de reconfiguração faz com que o FPGA seja flexível mantendo uma velocidade de processamento muito superior a de uma implementação equivalente em *software*. Essas características tornam o FPGA atrativo para as redes corporais.

2.1.1 Fluxo de projeto CAD

Para que seja possível a utilização prática do FPGA, são necessárias ferramentas que automatizem a maior parte do processo de desenvolvimento. Dessa forma o projetista deve se preocupar com a arquitetura do sistema enquanto a ferramenta CAD faz o trabalho de mapeamento dessa arquitetura para o FPGA. Nesta subseção serão resumidas as principais etapas, ilustradas na Figura 2.2, pelas quais necessariamente o fluxo de projeto deve passar. Neste fluxo não estão inclusas etapas como a simulação que, apesar de serem importantes, são opcionais.

O primeiro passo da implementação de um circuito em um FPGA é a **descrição desse circuito** em um uma linguagem de descrição de *hardware*, como o VHDL e o Verilog. Assim como nas linguagens de programação de alto nível para microprocessadores, essa descrição deve ser transformada em uma codificação que possa ser executada no *hardware* de destino. Este código deve então passar por um processo de tradução para uma representação em nível de portas lógicas básicas que representam funções booleanas simples. Essa representação simples pode passar por uma otimização em nível de lógica booleana independente de tecnologia visando minimizar as portas lógicas necessárias [22].

Esse processo de tradução é chamado de **síntese lógica**. Embora a síntese possa ter diferentes níveis dependendo da complexidade dos níveis entre os quais se quer transformar, a síntese lógica mais simples basicamente transforma uma representação de alto nível em uma lista de interconexões entre portas lógicas. Esse formato ainda pode ser representado em uma linguagem de descrição de *hardware* para que possam ser realizadas

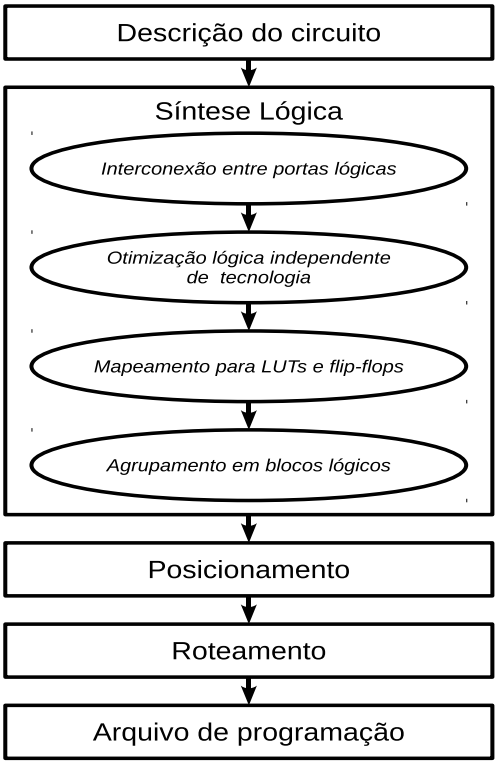


Figura 2.2: Etapas do fluxo CAD.

simulações pós-síntese. Geralmente, o formato gerado para a etapa seguinte é um formato próprio da empresa que desenvolve a ferramenta ou o formato EDIF [23] que é bastante popular.

Após a implementação de otimizações na rede booleana, chega ao final o processo de síntese independente da tecnologia. Na etapa seguinte, o **mapeamento de tecnologia**, mapeiam-se as portas lógicas para as LUTs e *flip-flops* no FPGA. Essa etapa é muito importante, pois a ferramenta deve saber aproveitar ao máximo a funcionalidade disponibilizada pelos componentes da tecnologia de destino. O resultado do mapeamento é uma lista de interconexões entre as LUTs e os *flip-flops*.

Em seguida, esse conjunto de LUTs e *flip-flops* é agrupado de acordo com o número de LUTs disponibilizadas em um bloco lógico do FPGA destino. Os algoritmos de agrupamento tiram vantagem das conexões internas de um bloco lógico que são mais rápidas que o roteamento de propósito geral. Ao final desta etapa, os blocos básicos estão agrupados em blocos lógicos, resultando em um conjunto de blocos lógicos e suas interconexões. Esses são mapeados para coordenadas de bloco lógico no FPGA de destino na etapa de posicionamento.

O **posicionamento** basicamente mapeia cada bloco lógico em uma posição fixa no FPGA. Esta etapa tem uma grande influência no desempenho do FPGA como um todo, portanto recebe uma atenção especial e muita pesquisa tem sido realizada para otimizá-la [24]. Um conjunto de blocos lógicos pode ser colocado em qualquer posição no FPGA e

o objetivo é buscar as posições ótimas para cada bloco. Além disso, uma vez posicionados os blocos precisam ser roteáveis, isto é, para cada interconexão necessária entre dois blocos quaisquer, devem existir linhas de roteamento suficientes para realizar todas as interconexões necessárias. Essas características fazem do posicionamento um problema NP-difícil [25]. Se o posicionamento é válido, ao final dessa etapa os blocos lógicos têm posição definida no FPGA e os elementos físicos de roteamento entre os blocos já podem ser definidos na próxima etapa.

Uma vez posicionados os blocos no FPGA, a etapa de **roteamento** define como os elementos serão conectados. Isso implica na escolha de onde cada interconexão lógica vai ser implementada fisicamente, escolhendo os segmentos, as trilhas, e os pinos de entrada e saída de cada bloco lógico que serão utilizados. Os algoritmos de roteamento também tentam minimizar a distância percorrida pelo sinal entre dois blocos lógicos, principalmente se a conexão pertence ao caminho crítico (conjunto de conexões que somam o maior atraso do circuito).

Terminado o roteamento, os *bits* de configuração do FPGA podem ser definidos e a programação do dispositivo de destino pode ser realizada. São definidos os *bits* que irão preencher as LUTs, os seletores dos multiplexadores, os controles dos *gates* dos transístores de passagem e dos *buffers tristate*. Esses *bits* formam o arquivo de configuração e geralmente são armazenados em células SRAM no FPGA.

2.1.2 Posicionamento em FPGAs

O problema de posicionamento em FPGAs é de difícil formalização matemática, pois não depende apenas da atribuição de posições dos blocos lógicos, mas também que eles sejam roteáveis [25]. De forma simplificada, seja $M = \{m_1, m_2, ..., m_n\}$ um conjunto de módulos, no qual um módulo m_i representa um bloco lógico que contém pinos de entrada e saída; e seja $L = \{l_1, l_2, ..., l_p\}$ uma representação das possíveis posições às quais um módulo pode ser atribuído, cada posição $l_i \in L$ pode ser representada através de coordenadas $l_i = (x_i, y_i)$. O problema de posicionamento pode ser resumido como a atribuição de um módulo $m_i \in M$ a uma localização $l_j \in L$ no FPGA de forma que o resultado satisfaça um determinado objetivo como, por exemplo, minimizar o atraso do caminho crítico.

Além de posicionar o circuito de forma a satisfazer uma função objetivo, devem existir segmentos no FPGA suficientes para que todas as conexões lógicas sejam implementadas fisicamente. Esses requisitos fazem com que o problema de posicionamento em FPGAs não possa ser resolvido em tempo polinomial. Existem, no entanto, algoritmos heurísticos que oferecem soluções aproximadas satisfatórias.

As técnicas de posicionamento com maior sucesso em FPGA utilizam heurísticas como a Têmpera Simulada (*Simulated Annealing - SA*) [26]. Um exemplo de algoritmo que utiliza SA é o algoritmo de posicionamento implementado no *Versatile Place and*

Route (VPR) [27]. Ele gera um posicionamento inicial aleatório e segue suas iterações realizando trocas de posição entre dois blocos quaisquer com o auxílio de uma variável de controle e da variação de custo entre o posicionamento anterior e o posicionamento atual. Inicialmente, o algoritmo permite trocas "ruins", ou seja, trocas que irão resultar em um custo maior. Ao passo que a variável de controle vai sendo decrementada, somente trocas que resultam em um custo menor são aceitas, embora ainda possa aceitar trocas "ruins". Na fase final do algoritmo, a variável de controle recebe o valor 0 e, a partir desse momento, nenhuma troca "ruim" é permitida.

Uma ferramenta de posicionamento muito utilizada no meio acadêmico é o VPR. Essa ferramenta alcança resultados bastante satisfatórios e geralmente é utilizada como referência para saber quão boa é uma nova solução de posicionamento e roteamento [25]. A versão original do VPR implementa algoritmos que tentam criar um compromisso entre o comprimento das conexões e o atraso do caminho crítico. Para tal, é utilizada a função de custo mostrada na Equação 2.1, na qual λ é a constante de compromisso entre atraso do caminho crítico e o comprimento das conexões, T_n é o custo atual de atraso do caminho crítico, T_{n-1} é o custo de atraso do caminho crítico anterior ($\Delta T = |T_n - T_{n-1}|$), W_n é o custo atual do comprimento das conexões e W_{n-1} é o custo de comprimento das conexões anteriores.

$$\Delta C = \lambda \, . \, (\frac{\Delta T}{T_{n-1}}) \; + \; (1 - \lambda) \, . \, (\frac{\Delta W}{W_{n-1}}) \tag{2.1}$$

Para desenvolver um posicionamento visando à redução do consumo de energia, Lamoureux et al. [9] altera a função de custo em sua implementação de um fluxo CAD ciente do consumo de energia, adicionando um compromisso entre consumo de energia e comprimento das conexões. A nova função de custo dessa versão alterada do VPR é mostrada na equação 2.2, na qual foram adicionados uma constante de compromisso entre consumo de energia e comprimento das conexões, σ, o custo atual da potência dissipada, P_n e o custo anterior da potência dissipada, P_{n-1}. O autor justifica que melhorar o consumo de energia não é útil se faz com que o atraso do caminho crítico aumente de forma significativa. Por isso, como se pode ver nessa equação, a fração correspondente a variação do custo de tempo não foi alterada.

$$\Delta C \;\; = \;\; \lambda \, . \, (\frac{\Delta T}{T_{n-1}}) \; + \; (1 - \lambda) \, . \, [\, (1 - \sigma) \, . \, (\frac{\Delta W}{W_{n-1}}) \; + \; \sigma \, . \, (\frac{\Delta P}{P_{n-1}}) \,] \tag{2.2}$$

Em um trabalho recente, El-Abd et al. [25] utilizam otimização por enxame de partículas para resolver o problema de posicionamento obtendo resultados próximos ao algoritmo utilizado no VPR para o tamanho da caixa delimitadora. Da mesma forma, considera-se

que outros algoritmos bio-inspirados também sejam adequados à resolução do problema.

2.1.3 Segmentação

A segmentação é uma técnica na qual operações podem ser sobrepostas para executarem concorrentemente [28]. A segmentação considera que as tarefas são compostas de etapas, muitas vezes comuns, que podem executar concorrentemente entre si. A segmentação de circuitos digitais é implementada usando registradores, que delimitam e armazenam os valores de cada etapa. O próximo parágrafo apresenta uma definição formal, generalizando o conceito de segmentação.

Seja um conjunto de tarefas $T = \{t_0, t_1, ..., t_n\}$, no qual a cada tarefa t_i está associado um tempo de execução de $\theta_i u$ (onde u é uma unidade de tempo qualquer). Por exemplo, a sequência de execução t_0, t_1, t_2 normalmente leva $(\theta_0 + \theta_1 + \theta_2)u$ para executar, pois a tarefa seguinte somente inicia após o término da anterior. Cada tarefa $t_i \in T$ é composta de uma sequência de etapas $e_{ij} | j \in \{0, 1, ..., m\}$ onde cada e_{ij} leva um tempo de $\psi_j u$ para finalizar. Pode-se notar que se n tarefas t_i, t_{i+1}, t_n forem compostas das mesmas etapas, ou seja $e_i j = e_{(i+1)j}$ $\forall$ i, j, para que duas etapas quaisquer e_{ij} e e_{kl} possam executar concorrentemente, basta que $j \neq l$. Por exemplo, dada a sequência de execução t_0, t_1, t_2, as sequências de etapas e_{02}, e_{11}, e_{20} e e_{03}, e_{12}, e_{21} podem ser executadas concorrentemente. Nos circuitos digitais a segmentação se torna mais complexa. Por exemplo, duas etapas diferentes podem usar os mesmo recursos de *hardware* de forma simultânea (memória e banco de registradores).

Nos circuitos digitais a segmentação é implementada através de registradores. Eles funcionam como uma memória entre uma etapa e outra, no entanto as operações de armazenamento ocorrem apenas nas bordas de subida ou descida de um sinal de sincronização (*clock*). Dessa forma, o valor armazenado se mantém estável em uma etapa até a chegada da próxima borda de *clock*, quando um novo valor é carregado. Como os valores devem estar estáveis na chegada da borda de *clock*, o *clock* possui um período de duração maior ou igual ao da etapa de maior latência do caminho de dados. Por isso, algumas técnicas foram desenvolvidas para balancear os atrasos entre as diferentes etapas de forma a minimizar o período de *clock*. Dentre estas técnicas está a retemporização.

2.1.4 Retemporização

A retemporização (*retiming*) nos FPGAs envolve o balanceamento do atraso dos sinais entre dois registradores quaisquer dentro do circuito. Sendo um circuito um conjunto de elementos combinacionais, interconexões e registradores (*flip-flops*), a representação pode ser um grafo direcionado no qual os nós são elementos combinacionais, as arestas direcionadas são interconexões entre esses elementos e os pesos das arestas representam a

quantidade de registradores presentes naquela interconexão. Em [29] os autores também apresentam um algoritmo para minimizar o período de *clock* usando retemporização. Esse processo envolve transformações da forma $w_r(e) = w(e) + r(v) - r(u)$, na qual o novo número de registradores na conexão e, $w_r(e)$, é resultado de se moverem $r(v)$ registradores para a conexão e e remover $r(u)$ registradores dessa conexão. Os autores provam que se $w_r(e) >= 0$ e se em todo ciclo no circuito existe ao menos um registrador a retemporização é válida. Em [30], Leiserson e Saxe dão uma prova mais detalhada de que as transformações $w_r(e)$ não alteram a funcionalidade do circuito, se as restrições forem respeitadas. A retemporização começa a ter sua utilidade para a economia de energia em FPGAs mais evidente quando se fala em *C-slow retiming* [31], uma técnica para fazer segmentação automaticamente, com prováveis reduções no *glitching*.

O *C-slow retiming* é uma técnica que permite criar registradores replicando C vezes os registradores existentes mantendo todos na mesma conexão do registrador original. Através das transformações realizadas pela retemporização descritas anteriormente, os registradores são redistribuídos pelo circuito balanceando os atrasos. Weaver et al. [31] implementam *C-slow* com C variando de 1 a 5 em um Virtex da Xilinx, mostrando os ganhos na frequência de *clock* e no número de tarefas processadas de forma concorrente.

O *C-slow retiming* pode, no entanto, alterar bastante as interconexões, pois o circuito já está posicionado e precisa-se encontrar registradores livres para efetuar as novas conexões. Singh e Brown [32] apontam alguns problemas nas abordagens utilizadas até então para realizar o retemporização. Segundo eles, o método mais comum aplica a retemporização na rede de portas lógicas antes do posicionamento. Como o principal componente dos atrasos é devido ao roteamento de propósito geral, não é possível obter informações precisas dos atrasos antes do posicionamento do circuito.

Singh e Brown também criticam a abordagem de retemporização após o posicionamento. A retemporização necessita movimentar os registradores ao longo das interconexões alterando sua estrutura. Se não houver registradores disponíveis na interconexão para a qual um registrador deve ser movido, deve-se realocar os circuitos de um modo que a adição de registradores naquela interconexão seja possível. Em [32], eles integram o posicionamento e a retemporização de forma que são realizadas as etapas de posicionamento, retemporização e uma fase posterior para ajustar as interconexões.

Eguro e Hauck [33], no entanto, criticam essa abordagem dividida em etapas em seu trabalho, pois a realização da retemporização como uma etapa isolada seguida de uma etapa de reajuste pode produzir um posicionamento ilegal ou um posicionamento legal com um atraso de caminho crítico maior. Apontam como solução a realização simultânea de posicionamento e retemporização utilizando *Simulated Annealing*, mostrando melhorias significativas em relação as demais abordagens em termos de comprimento das conexões e atraso do caminho crítico.

Apesar dos trabalhos anteriores mostrarem bons resultados com a realização com-

binada das fases de posicionamento e retemporização, a economia de energia não é o foco. Neste trabalho, pretende-se mostrar que a combinação de posicionamento e *C-slow retiming* pode ser usada para obter uma economia significativa de energia em FPGAs.

2.1.5 Modelo de energia

Para que uma ferramenta CAD possa calcular uma estimativa do consumo de energia de um circuito, o custo computacional do algoritmo deve ser razoável. Os simuladores de circuitos conhecidos por sua precisão, como o HSPICE, requerem muito tempo para realizar essa estimativa, não sendo adequados para uso em conjunto com as ferramentas de posicionamento e roteamento em FPGAs, por exemplo. Portanto, para que ferramentas CAD possam estimar o consumo de energia, deve haver uma forma de estimar a atividade de um circuito sem utilizar simulação.

Geralmente, estima-se o consumo de energia dinâmica no FPGA com o auxílio de informações sobre a atividade do circuito, que contém a frequência de transições de sinal para as diferentes interconexões no FPGA [34]. A potência dinâmica dissipada é estimada no VPR utilizando o modelo mostrado em [12] através da equação 2.3, na qual y representa um componente no circuito, tal como os segmentos de fio, multiplexadores e LUTs, C_y é a capacitância de y, D_y é a densidade de transição de y (número de transações por ciclo), V_{sp} é a *supply voltage*, V_{sw} é a *swing voltage* e f_{clk} é a frequência de *clock*.

$$Potencia\_Dinamica \; = \sum_{y \; \in \; Circuito} 0.5 \; . \; C_y \; . \; V_{sp} \; . \; V_{sw} \; . \; D_y \; . \; f_{clk} \tag{2.3}$$

As informações elétricas do FPGA são definidas em um arquivo que descreve a sua arquitetura em detalhes. Uma vez carregado pelo VPR, este gera a arquitetura de um FPGA com auxílio de várias estruturas de dados, principalmente um grafo de recursos de roteamento. Através dessas estruturas, é possível obter os detalhes de cada componente e fazer a estimativa da potência dissipada em cada um deles. A estimativa da atividade dos componentes pode ser obtida através de várias ferramentas, como por exemplo o ACE 2.0 de Lamoureux et al. [34]. Esta ferramenta apresenta bons resultados, próximos dos resultados obtidos por simulação.

O ACE recebe como entrada a saída do mapeamento de tecnologia, que contém LUTs e *flip-flops*, e gera as atividades. Após sua execução, uma versão modificada do T-VPack é usada para criar os *clusters* e gerar as atividades com as devidas modificações. A rede de blocos lógicos e os arquivos contendo as atividades dos diferentes componentes são então passados para o VPR para a realização do posicionamento e roteamento. Ao final, o modelo faz a estimativa do consumo de energia tomando como base as interconexões físicas e as estimativas de atividade.

2.2 Técnicas para otimização do consumo de energia

Os trabalhos que visam melhorar o FPGA geralmente focam em alguns dos ramos mostrados na Figura 2.3. Como um dos resultados desta pesquisa, foi criada uma taxonomia desses trabalhos nos três ramos: área, a velocidade e consumo de energia. Embora a área e a velocidade também sejam importantes, o maior desafio para a utilização dos FPGAs em dispositivos portáteis é o consumo de energia. Portanto, nesta subseção serão detalhados somente os trabalhos para a otimização do consumo de energia.

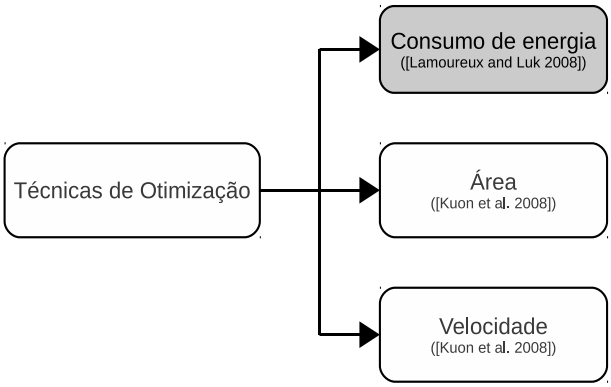


Figura 2.3: Técnicas de otimização.

As principais empresas desenvolvedoras de FPGAs estão buscando a redução no consumo de energia em seus produtos visto a crescente demanda por dispositivos que se comunicam por redes sem fio e usam fontes de energia limitadas, tais como baterias. Como exemplo de tais dispositivos, temos os nós de redes de sensores. Uma das possíveis organizações de uma rede de sensores é a organização em *clusters*. Nessa organização os sensores elegem um *clusterhead* que recebe os dados dos demais sensores, agrega os dados e envia os dados agregados para o *sink*.

Commuri et al. [5] desenvolveu em seu trabalho um *clusterhead* reconfigurável utilizando um FPGA. Esse *clusterhead* permite que os sensores pertencentes ao seu *cluster* comuniquem-se com ele utilizando baixa potência de transmissão de forma a economizar energia. Este realiza então a agregação de dados e comunica os dados agregados ao mundo externo. Através da reconfiguração do FPGA é possível implementar métodos de agregação de dados e protocolos de comunicação diferentes. Embora este trabalho mostre uma implementação eficiente utilizando FPGAs em relação a uma implementação em *software*, ele não leva em conta os gastos de energia no *clusterhead*. As principais técnicas para redução do consumo de energia em FPGAs são descritas nesta seção e ilustradas na Figura 2.4. Elas são apresentadas em quatro subseções seguindo a classificação adotada em [11]: nível de sistema, nível de arquitetura e nível de dispositivo e ferramentas CAD.

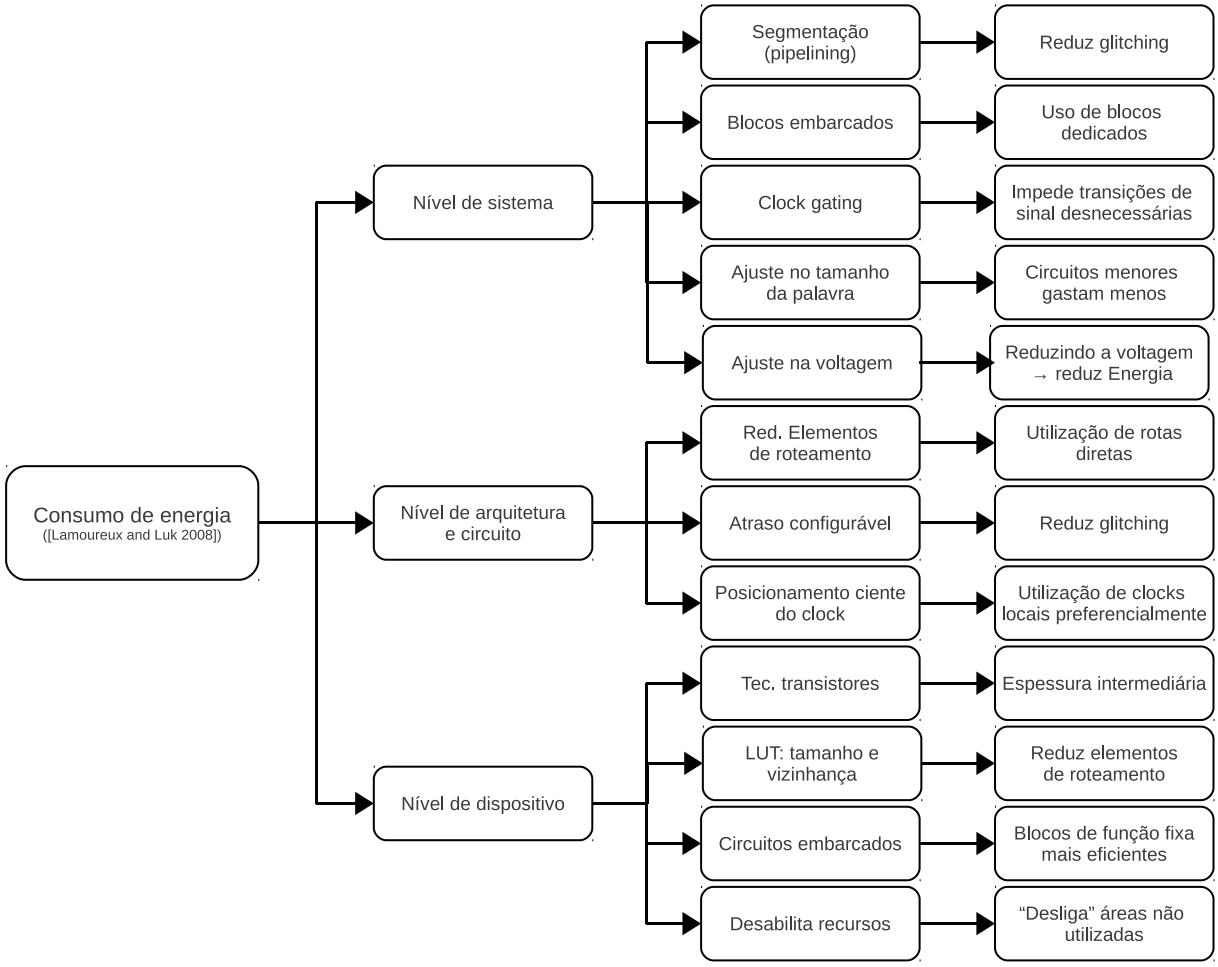


Figura 2.4: Técnicas para otimização do consumo de energia.

2.2.1 Nível de sistema

As técnicas de otimização do consumo de energia em nível de sistema podem ser decididas pelo projetista da aplicação que executa no FPGA [11]. Em nível de sistema, o projetista baseia-se nas informações sobre o *hardware* de destino para projetar um circuito com o objetivo de tirar vantagens dessas funcionalidades. Isso incentiva o uso de um circuito integrado embarcado de aplicação específica, de granularidade maior, para fazer multiplicações, por exemplo.

A energia dinâmica é aquela dissipada quando a aplicação está executando no FPGA, portanto tem relação direta com a frequência de transições dos sinais [12]. As alterações nas entradas se propagam pelo circuito causando oscilação nos sinais internos e terminando com uma possível alteração nos sinais de saída. Quando as entradas chegam aos elementos lógicos em tempos desiguais pode ocorrer o *glitching*.

O *glitching* é uma transição indesejada do sinal na saída de uma porta lógica, causada pelo atraso desigual de propagação entre suas entradas [13] . Ele pode ser reduzido através de segmentação, na qual as operações se dividem em estágios e dentro de cada estágio o sinal só pode prosseguir até o próximo registrador. Dessa forma, há uma chance maior

do sinal passar pelas operações intermediárias e estar estável nas entradas dos circuitos combinacionais intermediários.

2.2.2 Nível de dispositivo

Em nível de dispositivo encontram-se as técnicas desenvolvidas pelo projetista do circuito FPGA em si. Essas técnicas adaptam a tecnologia do circuito integrado para economizar energia. Uma das técnicas envolve a tecnologia dos transístores. Os transístores com portas menores são mais rápidos, mas gastam mais energia. Por outro lado portas maiores gastam menos energia e têm um desempenho inferior. Os principais fabricantes de FPGAs então optaram por desenvolver um tamanho intermediário, oferecendo um bom compromisso entre consumo de energia e desempenho.

O circuito de roteamento de propósito geral responde por grande parte do total da energia consumida no FPGA [11]. Algumas das técnicas para evitar o uso do roteamento de propósito geral já foram discutidas anteriormente, o aumento da granularidade do bloco através do aumento do número de entradas por LUT, o número de LUTs por bloco e ainda as ligações diretas entre blocos vizinhos.

Os fabricantes de FPGAs estão embarcando em seus produtos os circuitos que são mais utilizados pelas aplicações. Dessa forma, em vez de utilizar blocos lógicos para implementar um circuito comum, por exemplo um multiplicador, a aplicação pode utilizar o bloco embarcado. Assim, essas operações são otimizadas e os blocos lógicos reconfiguráveis ficam disponíveis para outras operações que necessitem realmente de sua flexibilidade.

2.2.3 Nível de arquitetura e circuito

O nível de arquitetura e circuito está relacionado com o projeto da arquitetura do FPGA e o mapeamento da aplicação nessa arquitetura. Como foi mencionado na subseção anterior, o circuito de roteamento de propósito geral deve ser evitado se possível. A arquitetura do FPGA deve disponibilizar formas de impedir transições de *clock* ou mesmo desligar partes do circuito não utilizadas. O trabalho proposto por Chow et al. [35] mostra um exemplo no qual portas *and* controlam as entradas dos circuitos, "desligando-os". De forma similar, pode-se desligar recursos de roteamento não utilizados, para economizar energia.

As transições de estado de *flip-flops* em regiões não ativas do circuito também ocasionam desperdício de energia nessas regiões, pois se o estado do *flip-flop* não é alterado, as transições de sinal nessa região não ocorrem. O *clock gating* reduz o esse consumo de energia dinâmica impedindo transições de estado dos *flip-flops*. Para implementar o *clock gating* em ASICs geralmente se faz um *and* do sinal de *clock* com um sinal de controle. Como os FPGAs têm fios dedicados para implementar as redes de *clock*, o sinal de *clock*

vai diretamente para os *flip-flops* sem a possibilidade de inserção de portas lógicas nesse caminho. Dessa forma, a implementação em FPGAs faz com que a saída do *flip-flop* seja opção de entrada para o mesmo [36]. Na próxima borda do *clock* o *flip-flop* recebe o mesmo valor e a transição de sinal não ocorre.

Lamoureux et al. [13] mostram a técnica do *delay* programável. Ela consiste na inserção de *delays* nos fios para que os sinais possam se estabilizar em tempos semelhantes e evitar transições de sinal indesejadas. Basicamente, o FPGA oferece a tecnologia necessária e o projetista utiliza as informações de atraso de propagação no circuito para saber qual sinal atrasar e em que parte do circuito. Realizando o mapeamento ciente desses recursos os sinais têm maior chance de se estabilizar nas entradas dos blocos lógicos, reduzindo o *glitching*.

Geralmente uma arquitetura complexa implementada em um FPGA envolve circuitos operando em diferentes frequências de *clock*. As fontes de *clock* podem ser externas ao FPGA, ou gerado por *phase-locked loops (PLLs)* ou *delay-locked-loops (DLLs)* internos e se distribuir dentro do mesmo por redes de *clocks* globais ou locais, como mostrado em [37]. Em [38], Lamoureux et al. propõem o posicionamento ciente de *clock* para reduzir os caminhos entre a fonte de *clock* e os circuitos que os recebem.

2.2.4 Ferramentas CAD

As ferramentas CAD, utilizadas nos projetos das aplicações para FPGAs, implementam internamente algoritmos que visam economizar energia. Como exemplo, a versão do VPR modificada por [39, 9] implementam internamente o modelo desenvolvido por Poon et al. [12] para estimar o consumo de energia em FPGAs. Além de estimar o consumo de energia de um circuito qualquer, podem-se desenvolver técnicas que utilizam o modelo para alterar o projeto de forma que ele seja mais econômico em termos de energia.

Lamoureux et al. [9] implementam algoritmos para o mapeamento de tecnologia, a formação de *clusters*, o posicionamento e o roteamento voltados à economia de energia. Eles utilizam o modelo de economia de energia citado anteriormente para estimar o consumo de energia de cada alternativa de projeto. Eles mostram, ainda, reduções significativas no consumo de energia, principalmente com mapeamento de tecnologia e formação de *clusters*.

2.3 Inteligência artificial

Segundo [40], a inteligência artificial pode ser definida como o estudo de ideias que visam trazer para as máquinas a capacidade de raciocínio e comportamento semelhantes aos dos humanos. Deste estudo, nasceu um campo da ciência da computação que visa prover inteligência à entidades artificiais, geralmente trechos de *software* denominados de

"agentes". Esses agentes possuem comportamento e raciocínio que os tornam capazes de resolver determinados problemas. Nos métodos tradicionais de inteligência artificial, a inteligência existe a priori de forma centralizada e é embutida na máquina.

Uma outra forma de inteligência artificial é obtida através da computação bio-inspirada. Nesta, busca-se inspiração em mecanismos que ocorrem na natureza, onde sistemas naturais resolvem problemas bastante complexos para manter seu funcionamento. Um exemplo é o organismo humano, que resolve problemas bastante complexos através de sistemas como o sistema imunológico (defesa), o sistema nervoso (controle) e o sistema endócrino (comunicação) [41, 42, 43]. A computação bio-inspirada difere da inteligência artificial tradicional em alguns pontos, pois a inteligência surge de forma distribuída através de interações entre os componentes.

Alguns dos algoritmos bio-inspirados são fundamentados no funcionamento dos processos relacionados ao genoma em uma célula ou na interação entre células em organismos multicelulares. Geralmente as características que mais atraem nesses sistemas são sua capacidade de auto-organização e sua adaptabilidade a mudanças no ambiente. Os algoritmos genéticos se baseiam no funcionamento da evolução dos seres vivos através de modificações sucessivas no seu genoma no decorrer dos anos [44]. Mapeado para a computação, na primeira geração um cromossomo ou um conjunto de cromossomos representa solução inicial do problema. Geração após geração, esta solução inicial é melhorada através de processos de mutação ou recombinação. Esse processo se repete até que uma solução satisfatória seja encontrada ou o número máximo de gerações seja produzido.

As redes neurais são outra forma bio-inspirada de resolução de problemas [43]. Estas são inspiradas nos neurônios biológicos e as interações que fazem com que eles sejam capazes de resolver diversos problemas. Para isso, foram desenvolvidos diversos modelos de neurônio e interações sinápticas de forma a simular o comportamento de neurônios biológicos. Esses modelos consistem basicamente de neurônios artificiais cujos dendritos geram um estímulo resultante de uma função ou peso. Esta função ou peso é alterada conforme a rede "aprende", causando uma adaptação às condições atuais. O neurônio recebendo os sinais ponderados dos diferentes dendritos, verifica se a computação de uma determinada função passa do limite definido para que o seu sinal de saída seja lançado através do seu axônio artificial.

Para acelerar tais algoritmos, foram desenvolvidas abordagens em *hardware*, dentre elas os projetos POEtic [45, 46, 44, 47, 42] e Perplexus [48]. Estes visam criar um organismo a partir de uma célula inicial (zigoto) na qual podem ser aplicados mecanismos evolutivos, como por exemplo, algoritmos genéticos. Esta célula se divide e as novas células adquirem funcionalidades diferentes dependendo de sua posição (diferenciação). Uma vez que o organismo já "cresceu" o suficiente, ele começa a interagir com o meio e adaptar sua funcionalidade a ele. Como um exemplo prático desse momento do desenvolvimento do organismo, cada célula pode representar um neurônio e o organismo ser uma

rede neural. O objetivo desse ambiente era dar suporte ao desenvolvimento de "organismos" em hardware que fossem capazes de auto-organização, auto-reparo e adaptação ao meio, assim como os organismos biológicos multicelulares.

Outra forma de bio-inspiração na natureza se relaciona a população de organismos. Nesta, o foco não é a estrutura interna de um um único organismo, mas o seu comportamento individual e sua interação com os demais organismos para resolver um determinado problema. Este assunto será detalhado na subseção a seguir.

2.3.1 Inteligência de enxames

A inteligência de enxames é um campo da inteligência artificial que busca entender o comportamento de indivíduos sociais na natureza descobrindo a inteligência que emerge dessa organização coletiva [49]. Alguns indivíduos, como exemplos notáveis as abelhas e as formigas, formam colônias nas quais alguns subgrupos são responsáveis por tarefas específicas e às vezes são morfologicamente diferentes dos demais. Cada indivíduo realiza tarefas simples cooperando com os demais.

Um exemplo simples de cooperação entre os indivíduos em uma colônia pode ser visto em uma colônia de formigas. Uma vez que uma formiga encontra o alimento, ela anda deixando uma trilha de feromônio por onde passa. Outra formiga que esteja por perto e encontre essa trilha, passa a seguir por ela reforçando-a com seu próprio feromônio. A constante operação de várias formigas faz com que uma trilha seja formada com uma boa alternativa de caminho entre o ninho e a fonte de alimento.

Um indivíduo, ou seja, uma formiga, tem como objetivo apenas encontrar alimento ou uma trilha de feromônio. O trabalho em conjunto, por outro lado, faz com que um bom caminho seja formado entre o ninho e a fonte de alimento, fazendo com que as formigas alimentem o formigueiro de uma forma mais rápida e eficiente. Assim como a cooperação de indivíduos simples levou à resolução de um problema mais complexo, os sistemas computacionais podem se inspirar nessa forma de cooperação para resolver os problemas de natureza semelhante. Dessa forma, inspirados nesses comportamentos coletivos, foram desenvolvidos algoritmos que buscam trazer essa auto-organização para resolver problemas computacionais que geralmente não podem ter sua solução exata determinada em tempo polinomial.

A inteligência de enxames têm sido utilizada com sucesso, oferecendo soluções satisfatórias para problemas NP-completos. Um dos algoritmos mais conhecidos nessa área é a otimização por colônia de formigas [14]. O algoritmo encontra o melhor caminho entre dois pontos utilizando agentes móveis inspirados em formigas e sua forma de criar uma rota do ninho até o seu alimento.

Deneubourg et al. [50] observaram que em uma colônia de formigas os indivíduos têm a tendência de carregar um objeto se este estiver isolado e de colocá-lo onde já

existem outros do mesmo objeto. Eles desenvolvem um modelo matemático para simular o comportamento desses indivíduos utilizando funções probabilísticas. Nesse modelo, os indivíduos artificiais andam aleatoriamente e pegam ou deixam um objeto respeitando as equações 2.4 e 2.5. Nos seus experimentos, tais indivíduos têm um comportamento de agregação de objetos muito próximo ao das formigas nas quais são inspirados.

$$P_{pegar} = (\frac{k_p}{k_p + f})^2 \tag{2.4}$$

$$P_{deixar} = (\frac{f}{k_d + f})^2 \tag{2.5}$$

Nas equações 2.4, P_{pegar} é a probabilidade de um agente pegar um objeto. Na equação 2.5, P_{deixar} é a probabilidade do agente deixar objeto que está carregando no local onde ele se encontra atualmente. As constantes k_p e k_d são ajustadas para facilitar ou dificultar as ações de pegar e deixar, respectivamente. A equação também apresenta a função f que define o grau de similaridade entre os objetos e guia as decisões dos agentes. Quanto maior for o valor de f maior é a chance de um agente deixar o objeto e menor a chance de retirar o objeto daquele local.

2.4 Conclusão

Este capítulo mostrou os fundamentos sobre os *Field-Programmable Gate Arrays (FPGAs)*. Enfatizou-se a possibilidade de configuração dos seus blocos lógicos e das suas conexões no roteamento para a obtenção de qualquer funcionalidade desejada. Foi também apresentada uma análise das etapas necessárias para se obter um circuito funcional em um FPGA a partir de uma linguagem de descrição de *hardware* e dos trabalhos que tiveram como objetivo otimizar essas etapas.

Dentre os trabalhos que atuam no fluxo CAD dos FPGAs, muitos focam na redução do consumo de energia. No entanto, a segmentação realizada antes do posicionamento leva a diferenças muito grandes nos atrasos de interconexões. Por outro lado, os trabalhos que focam na retemporização não apresentam segmentação automática ou o objetivo de reduzir o consumo de energia.

Por fim, este capítulo apresentou fundamentos e trabalhos na área de inteligência artificial. Ele mostrou como a inteligência artificial tem sido usada para resolver problemas cujas soluções determinísticas não apresentam um algoritmo de complexidade polinomial. Um importante campo da inteligência artificial, a inteligência de enxames, traz da natureza soluções para problemas complexos através de unidades de trabalho relativamente simples, sendo a inspiração em colônias de formigas um dos principais exemplos.

CAPÍTULO 3

A TÉCNICA BIO-INSPIRADA ANTES PARA ECONOMIA DE ENERGIA EM FPGAS

Este capítulo apresenta uma técnica de posicionamento, segmentação e retemporização visando a redução do consumo de energia em FPGAs. A Seção 3.1 mostra uma visão geral da técnica, destacando seu escopo dentro do fluxo CAD. A técnica emprega agentes bio-inspirados cujas funções são descritas na Seção 3.2. Os algoritmos de posicionamento, e de segmentação e retemporização são descritos nas Seções 3.3 e 3.4.

3.1 Visão geral

A técnica proposta, chamada de **AntES** (*pl**A**cement, pipeli**N**ing and re**T**iming for **E**nergy **S**aving*), agrega as fases de posicionamento, segmentação e retemporização no fluxo CAD com o objetivo economizar energia em FPGAs. A **AntES** atua na etapa de posicionamento do fluxo CAD original, adicionando novas funções. Esta nova etapa de posicionamento consiste de uma fase inicial de segmentação seguida das fases de posicionamento e retemporização que ocorrem de forma intercalada. A Figura 3.1 ilustra a nova organização do fluxo CAD com o uso dessa técnica.

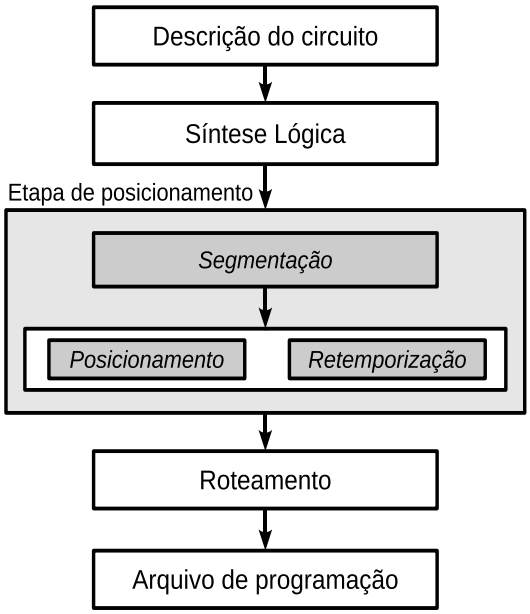


Figura 3.1: Segmentação, posicionamento e retemporização no fluxo CAD.

A fase de posicionamento da técnica **AntES** economiza energia ao reduzir o comprimento das conexões mais ativas através da aproximação dos blocos lógicos de origem e

destino dessas conexões. As fases de segmentação e retemporização pretendem reduzir as transições indesejadas dos sinais do FPGA, diminuindo o atraso entre os registradores no mesmo caminho. Como as transições de sinal estão diretamente relacionadas com a potência dinâmica dissipada, reduzir o número de transições diminui o consumo de energia.

A técnica **AntES** realiza segmentação, seguida de execução simultânea de posicionamento e retemporização. Para isso, ela emprega uma abordagem inspirada no comportamento de colônias de formigas, na qual agentes, isto é, trecho de código em *software*, movimentam os blocos lógicos, blocos de entrada e saída e os registradores pelo FPGA. Os agentes possibilitam que os movimentos individuais de posicionamento sejam intercalados com os de retemporização, pois eles agem de forma independente.

O modelo dos agentes baseados em colônia de formigas [50] utiliza funções de probabilidade que indicam quando um objeto deve ser retirado ou colocado em uma determinada posição. O principal componente desse modelo é a função de atratividade que informa o quanto uma posição é atrativa a um objeto. No modelo original, a atratividade de uma posição depende do número de objetos do mesmo tipo que ela contém. Dessa forma, quanto maior o número de objetos do mesmo tipo que uma posição contém, mais forte é a atratividade ("odor") desse tipo de objeto no local.

Na técnica **AntES**, os objetos movimentados são os registradores, os blocos lógicos e os bloco de entrada e saída. Dessa forma, a função de atratividade reflete a qualidade das conexões locais de um bloco ou o atraso entre registradores, dependendo do objetivo do agente. Assim, o "odor" é mais forte quando a qualidade das conexões locais ou o atraso entre registradores se aproximam do seu valor ótimo.

3.2 Agentes bio-inspirados

A **AntES** define três tipos de agentes: os agentes dos blocos lógicos (A_{lb}), os agentes dos blocos de entrada e saída (A_{iob}) e o agente de retemporização (A_{ret}). Os agentes são trechos de código em *software* que simulam o comportamento de uma formiga agrupando objetos na natureza. A Figura 3.2 mostra uma lista de interconexões entre quatro blocos lógicos e o posicionamento delas pelos agentes em uma matriz 3x3 de blocos lógicos, que representa a estrutura física de um FPGA. O agente A_{lb} é responsável por associar os blocos lógicos à posições nessa matriz de blocos do FPGA.

A Figura 3.3 mostra detalhes de um bloco básico e o escopo do agente A_{ret}. Ela ilustra a estrutura interna de um bloco lógico composta de dois blocos básicos. Cada bloco básico contém em seu interior um registrador. Este registrador é o componente de interesse dos agentes A_{ret}. Cada agente A_{ret} cria e remove registradores conforme efetua os movimentos de retemporização (discutidos mais detalhadamente nas próximas subseções). Durante a etapa de posicionamento, os agentes A_{lb}, A_{iob} e A_{ret} trabalham simultaneamente, cada

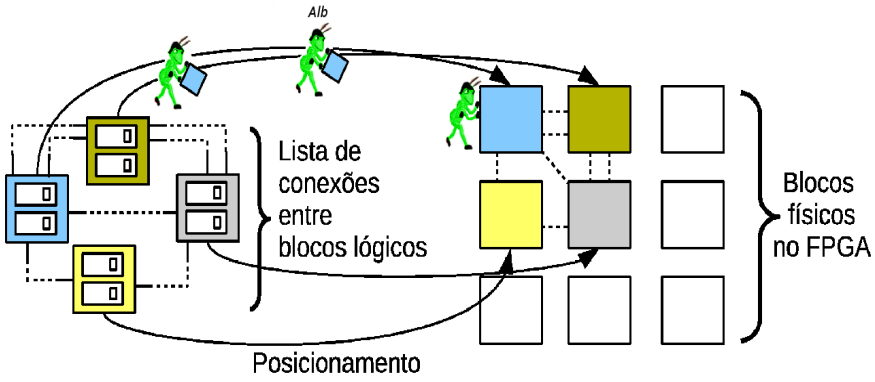


Figura 3.2: Posicionamento de dos blocos lógicos no FPGA empregando o agente A_{lb}.

um dentro do seu escopo de atuação.

Os agentes A_{lb} e A_{iob} são os agentes de posicionamento, sendo suas tarefas e objetivos similares. O objetivo de ambos é reduzir a distância entre blocos lógicos cuja interconexão tem grande número de transições. Por outro lado, o agente A_{ret} está associado à segmentação e à retemporização do circuito. Ele movimenta somente registradores e seu objetivo é balancear os atrasos entre os registradores do circuito. Para alcançar tal objetivo, o agente A_{ret} se desloca pelas interconexões e movimenta os registradores para frente ou para trás das LUTs caso considere significativo o atraso do caminho percorrido.

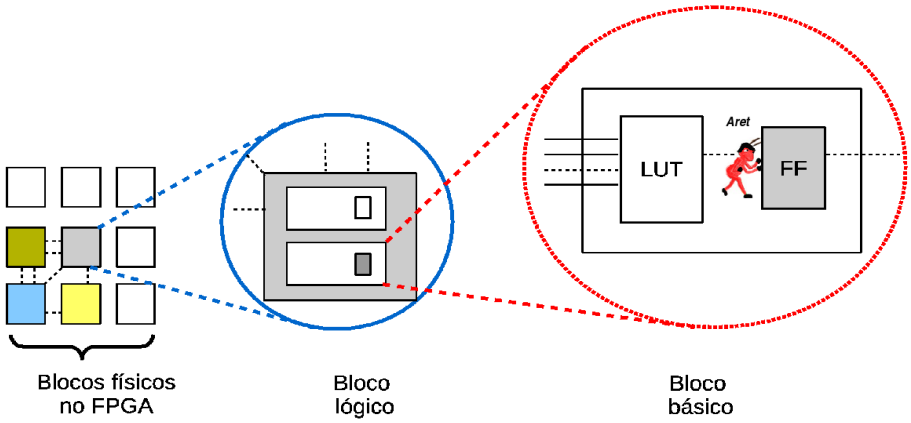


Figura 3.3: Detalhes de um bloco básico mostrando o escopo de atuação do agente A_{ret}.

O comportamento dos agentes é representado através das funções de probabilidade mostradas nas equações 3.1 e 3.2. Sempre que um agente retira um objeto (registrador, bloco lógico ou blocos de entrada e saída) de uma posição e se desloca para outra posição, ele verifica a probabilidade de deixá-lo na nova posição, $pDeixar$. Se por outro lado, o agente não está associado a um objeto, assim que ele chega a uma coordenada na qual há um bloco lógico posicionado, ele verifica a probabilidade de pegá-lo, $pPegar$.

$$pPegar = (\frac{k_p}{k_p + f})^2 \tag{3.1}$$

$$pDeixar = (\frac{f}{k_d + f})^2 \tag{3.2}$$

As constantes k_p, de "pegar", e k_d, de "deixar", são ajustadas para dificultar ou facilitar as operações de "pegar" e "deixar" um objeto, respectivamente. A variável f resulta de uma função de atratividade, representando o quanto uma posição é atrativa para um objeto, esteja ele já associado àquela posição ou associado a um agente. Quanto mais "atrativa" for uma posição, é mais fácil um agente deixar um objeto naquela posição e mais difícil de um agente retirá-lo daquela posição.

Os agentes A_{lb} associam os blocos lógicos a uma coordenada de bloco no FPGA. De forma similar os agentes A_{iob} posicionam os blocos de I/O. Os agentes A_{ret} retemporizam o circuito através da movimentação dos registradores ativos. Os três tipos de agentes têm frequência de movimentação diferentes, a frequência f_{lb} de movimentação dos agentes A_{lb} é maior que a frequência f_{iob} de movimentação dos agentes A_{iob}. Isso é feito para que a posição dos circuitos nos blocos lógicos tenha mais tempo para se adequar às posições das entradas e saídas nos blocos de I/O. A frequência f_{ret} é ajustada de acordo com a frequência de movimentação dos outros agentes.

3.3 Posicionamento usando agentes bio-inspirados

As etapas de mapeamento de tecnologia e agrupamento transformam uma rede de portas lógicas em uma lista de blocos e suas interconexões. O posicionamento desses blocos em coordenadas do FPGA é efetuado usando os agentes de posicionamento A_{lb} e A_{iob}. Os agentes A_{lb} se movem pelos blocos lógicos, enquanto os A_{iob} se movem pelos blocos de I/O. Inicialmente os blocos lógicos do circuito são posicionados em coordenadas arbitrárias da matriz de blocos lógicos do FPGA. Da mesma forma, os blocos de I/O do circuito são associados arbitrariamente aos blocos de I/O do FPGA. Ambos os agentes se movimentam sobre os blocos do seu tipo, ou seja, I/O ou lógico, e empregam as probabilidades $pPegar$, definida na Equação 3.1, para decidir se pegam um bloco, ou $pDeixar$, definida na Equação 3.2, para decidir se deixam um bloco em uma coordenada do FPGA.

Para decidir se pegam ou deixam, os agentes precisam ser capazes de analisar a qualidade do posicionamento atual de um bloco lógico. Para isso, é utilizada uma função de atratividade que representa o quanto uma posição é atrativa para um determinado bloco. Essa função tem como resultado f, utilizado para calcular $pPegar$ e $pDeixar$, influenciando diretamente nas decisões dos agentes.

Neste trabalho a função de atratividade leva em conta o consumo de energia das interconexões locais para que os blocos lógicos sejam posicionados de forma a reduzir esse consumo. A Equação 3.3 mostra o cálculo da função de atratividade A, no qual $f = A(P, T, W)$. Desse modo, seja b_{at} o bloco lógico atual associado a um agente ag, P então é a potência dissipada considerando as conexões locais do bloco b_{at}, T é o custo atual do atraso das conexões locais em b_{at}, e W é o comprimento das conexões locais em b_{at}. Para cada um desses parâmetros da função A, calcula-se uma proporção φ considerando seus valores máximos e mínimos. Além disso, de forma similar à equação de Lamoureux et al. (Equação 2.2 - Cap 2), são necessários termos de compromisso entre P, T e W para se chegar a um único valor com o objetivo de definir quais parâmetros têm maior prioridade sobre os demais. Assim, as constantes de compromisso λ e σ são definidas conforme a importância relativa de cada um dos parâmetros.

$$\varphi T = \frac{T - T_{min}}{T_{max} - T_{min}} \; ; \quad \varphi W = \frac{W - W_{min}}{W_{max} - W_{min}} \; ; \quad \varphi P = \frac{P - P_{min}}{P_{max} - P_{min}}$$

$$\varphi V = \lambda \; . \; (\varphi T) \; + \; (1 - \lambda) \; . \; [\; (1 - \sigma) \; . \; (\varphi W) \; + \; \sigma \; . \; (\varphi P) \;]$$

$$f = 1 - (\varphi V) \tag{3.3}$$

Embora a Equação 3.3 seja eficaz na determinação da probabilidade de um agente pegar um bloco, $pPegar(ag)$ (empregada no Algoritmo 1), a probabilidade de deixar um bloco b_{at}, $pDeixar(ag)$, é mais eficiente quando se leva em conta os custos da posição anterior do bloco. Isso acontece porque um agente ag não associado a um bloco lógico não dispõe de informação prévia sobre um bloco b_{at} que está posicionado na coordenada em que ele acabou de chegar. Para verificar a necessidade de retirá-lo dessa posição, o agente deve analisar os custos locais em relação aos valores extremos que P, T e W podem assumir. Já um agente ag associado a um bloco b_{at} conhece os custos da posição anterior desse bloco, e pode posicioná-lo em uma coordenada onde esses custos sejam menores, permitindo uma melhora progressiva.

A Equação 3.4 representa essa nova abordagem para a função de atratividade, denominada função A', na qual $f' = A'(P, T, W)$. Nela, o custo calculado por um dado agente ag para a posição atual do bloco lógico b_{at} é denotado C_n, onde $C \in \{T, P, W\}$. Da mesma forma, C_{n-1} representa o custo para a posição anterior. Assim como na Equação 3.3, a variação ΔC $(C_n - C_{n-1})$, ponderada com o auxílio das constantes de compromisso λ e σ, gera um valor entre 0 e 1, que representa a atratividade da posição atual para o bloco b_{at}. Com esse novo cálculo de f para $pDeixar(ag)$, o agente tem alta probabilidade de posicionar o bloco b_{at} quando a localização atual é melhor do que a anterior. Se a variação ΔV resultar em $zero$, a atratividade assume o valor 0.5, levando em conta que se o custo

não variou o movimento é facultativo.

$$\Delta T = \frac{T_n - T_{n-1}}{T_{n-1}} \; ; \quad \Delta W = \frac{W_n - W_{n-1}}{W_{n-1}} \; ; \quad \Delta P = \frac{P_n - P_{n-1}}{P_{n-1}}$$

$$\Delta V = \lambda \; . \; (\Delta T) \; + \; (1 - \lambda) \; . \; [\; (1 - \sigma) \; . \; (\Delta W) \; + \; \sigma \; . \; (\Delta P) \;]$$

$$f' = \begin{cases} 0.0 & \text{se } \Delta T \geq 1.0 \text{ ou } \Delta W \geq 1.0 \text{ ou } \Delta P \geq 1.0, \\ 1 - (\frac{\Delta V + 1}{2}) & \text{caso contrário.} \end{cases} \tag{3.4}$$

O Algoritmo 1 descreve a operação dos agentes A_{lb} e A_{iob} na definição do posicionamento. Inicialmente, os agentes são gerados e associados a coordenadas de bloco no FPGA ($l.2$). Cada agente é gerado em uma determinada coordenada (x, y) com uma probabilidade P_{gen}, como em [51]. Um tempo de vida é atribuído a cada agente de posicionamento, tal que se o seu tempo se esgota e ele não possui nenhum circuito associado, ele deixa de existir. Desse modo, seja M_x o número de blocos lógicos por linha da matriz de blocos lógicos do FPGA, M_y o número de blocos lógicos por coluna dessa matriz e M_{io} o número de blocos de I/O do FPGA. Um tempo de vida $t \in \mathbb{N}$ entre $(M_x + M_y)$ e $(M_x * M_y)$ é atribuído a cada agente A_{lb} e entre $\frac{M_{io}}{4}$ e M_{io} a cada agente A_{iob}. Isso possibilita aos agentes uma cobertura satisfatória dos blocos lógicos do FPGA.

O processo de posicionamento acontece enquanto existirem agentes associados a alguma coordenada no FPGA ($l.3$). Um agente que não possua um bloco, mas exista um bloco associado a sua posição atual, captura esse bloco com probabilidade $pPegar(ag)$ ($l.5$-9). Contudo, um agente possuindo um bloco, posiciona esse bloco na coordenada atual com probabilidade $pPdeixar(ag)$ ($l.10$-16). Se a posição atual estiver ocupada é realizada uma troca de posições e o bloco local é posicionado na coordenada anterior do bloco associado ao agente. Após decidir efetuar ou não as operações de pegar ou deixar, cada agente calcula sua nova posição ($l.17$). Quando os agentes tiverem o seu tempo de vida esgotado, uma avaliação global é realizada para determinar se o posicionamento atual é satisfatório ($l.20$).

O procedimento $calcNovaPos()$ ($l.22$) descreve o processo do cálculo de uma nova posição para os agentes A_{lb} e A_{iob}. Para acelerar a convergência na tarefa de posicionamento são mantidos históricos globais e por regiões que armazenam as direções recentemente tomadas pelos agentes. Entre o histórico global e o local, o agente escolhe em um determinado momento aquele que proporcionou o melhor resultado. O agente escolhe a direção do seu movimento ($l.23$-24) com base nesse histórico, tendo maior possibilidade de seguir a direção mais tomada. Cada vez que um agente se movimenta um contador de

Algoritmo 1 Processo de posicionamento.

```
1:  procedimento POSICIONAMENTO(Agentes, FPGA)   ▷ Agentes é uma lista que contém todos os agentes e FPGA é a matriz de blocos lógicos que representa o FPGA
2:      geraAgentes()
3:      enquanto Agentes ≠ ∅ faça   ▷ Enquanto existirem agentes, ou seja, enquanto o conjunto Agentes não estiver vazio
4:          para todos ag ∈ Agentes faça
5:              se ag.circuito = NULL então
6:                  se (pPegar(ag) ≥ r) and (FPGA(x, y).circuito ≠ NULL) então
7:                      ag.circuito ← FPGA(x, y).circuito   ▷ Associa o circuito com o agente
8:                      FPGA(x, y).circuito ← NULL
9:                  fim se
10:             senão
11:                 r ← rand()
12:                 se (pDeixar(ag) ≥ r) and (FPGA(x, y).circuito = NULL) então
13:                     FPGA(x, y) ← ag.circuito   ▷ Associa o circuito com uma posição no FPGA
14:                     ag.circuito ← NULL
15:                 fim se
16:             fim se
17:             calcNovaPos(ag)   ▷ Calcula uma nova posição para o agente
18:         fim para
19:     fim enquanto
20:     avaliaPosicionamento()
21: fim procedimento
22: procedimento CALCNOVAPOS(ag)
23:     dir ← escolheDirecao(contadorDirTomadas)   ▷ dir ∈ {direita, esquerda, cima, baixo}
24:     atualizaHistorico(ag.pos, dir)
25:     ag.saltos ← ag.saltos + 1
26:     ag.pos ← calcPos(dir, ag.pos, 2^ag.saltos)   ▷ ag.saltos é o número de saltos sem operações
27: fim procedimento
```

saltos é incrementado ($l.25$). Usando esse contador, a distância do "salto" aumenta em escala logarítmica fazendo com que o agente se distancie mais das regiões nas quais as alterações são desnecessárias ($l.26$).

3.3.1 Exemplo de posicionamento

A Figura 3.4 ilustra o posicionamento de duas portas lógicas, uma *and* e uma *ou-exclusivo*, efetuado pelos agentes A_{lb}. Neste caso, o FPGA contém 9 blocos lógicos organizados em uma matriz 3x3 circundada por 12 blocos de I/O. Como nas abordagens utilizando Arrefecimento Simulado (*Simulated Annealing - SA*), inicialmente se posiciona o circuito aleatoriamente. Nessa figura a posição inicial da porta *and* é o bloco lógico de coordenada (2,1) e da porta *ou-exclusivo* é o bloco lógico de coordenada (0,2). Essas duas posições possuem probabilidade P_{gen} de receber um agente. Neste exemplo foram gerados dois agentes A_{lb}, A e B, que foram associados às posições (0,2) e (2,1), respectivamente.

Uma vez na posição (2,1), o agente B verifica a probabilidade de pegar a porta *and* através da função $pPegar()$. A função de atratividade A, embutida em $pPegar()$, aponta uma baixa atratividade da posição (2,1) com relação a porta *and*, pois nesta posição esta

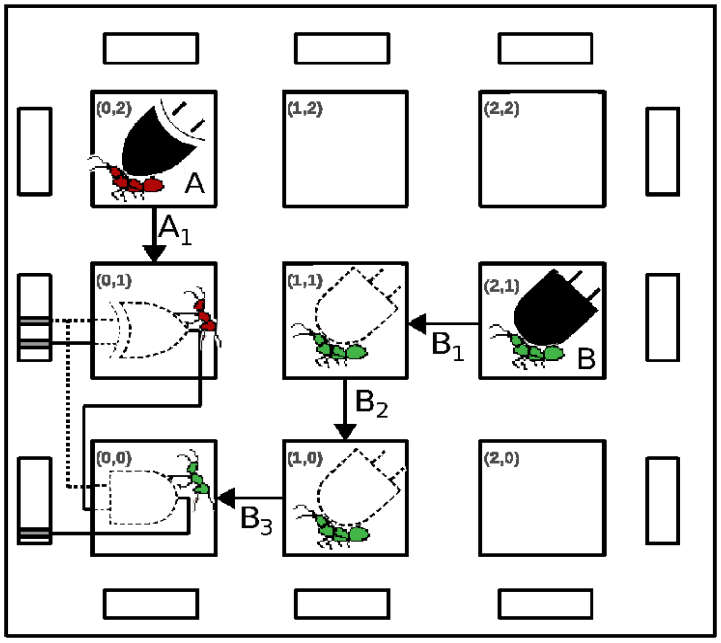


Figura 3.4: Posicionamento de um circuito composto por uma porta *and* e uma porta *ou-exclusivo* utilizando dois agentes.

porta ficaria distante dos outros componentes do circuito. Assim, o agente B retira a porta *and* da posição (2,1) e calcula sua próxima posição através da função $calcNovaPos()$. A nova posição calculada pelo agente B é a coordenada (1,1) e o agente se desloca para essa coordenada (movimento B_1). Nessa nova posição, o agente B verifica a probabilidade de deixar a porta *and*, através da função $pDeixar()$. Embutida em $pDeixar()$, a função de atratividade A' indica se a posição (1,1) é melhor que a posição (2,1) para deixar a porta *and*. O agente B decide não deixar a porta *and* na coordenada (1,1) e procura uma nova posição. Essas etapas se repetem até que o agente B chega à coordenada (0,0), na qual posiciona a porta *and* em definitivo. De forma similar, o agente A posiciona a porta *ou-exclusivo* na posição (0,1), após o movimento A_1. Na figura foram omitidos os agentes A_{iob}, no entanto, os três tipos de agentes, isto é A_{lb}, A_{iob} e A_{ret}, agem ao mesmo tempo.

3.4 Segmentação e retemporização usando agentes bio-inspirados

A segmentação geralmente é feita durante a fase de projeto da arquitetura do circuito. Desse modo, a redução do consumo de energia que poderia ser obtida pela segmentação é dependente do projetista da arquitetura. A técnica **AntES** realiza uma segmentação automática, ou seja, independente do projeto da arquitetura do circuito. Para isso, são adicionados registradores às entradas e saídas do circuito, pois estes não alteram sua funcionalidade e tornam a segmentação independente do número de registradores já existentes.

Nesta fase da técnica, esses registradores são replicados C vezes, em um processo similar ao *C-Slow* [31]. As etapas de adição, replicação e posterior retemporização dos registradores permitem que o circuito seja segmentado de forma automática por uma ferramenta CAD.

Essa forma de segmentação exige a retemporização do circuito para posicionar corretamente os novos registradores. A retemporização é realizada pelos agentes A_{ret}, e seu objetivo é balancear os atrasos das conexões entre os registradores do circuito, minimizando o período de *clock*. A retemporização reposiciona os registradores adicionados, "corrigindo" a fase de segmentação.

Os agentes A_{ret} são gerados e associados aos registradores previamente adicionados ao circuito, sendo que cada registrador recebe um agente A_{ret} com uma probabilidade P_{gen}. Logo, se N_R é o número de registradores adicionados às entradas e saídas do circuito e M_{ret} é o número de agentes A_{ret} gerados, então $M_{ret} \leq N_R * C$.

Uma das diferenças na movimentação dos agentes A_{ret} em relação aos agentes de posicionamento é a menor flexibilidade de movimento. Enquanto os agentes A_{lb} e A_{iob} podem se movimentar em qualquer direção independentemente das ligações entre os blocos, os agentes A_{ret} só se movimentam sobre as interconexões que partem do bloco básico ao qual estão associados atualmente. Para avaliar a necessidade de efetuar um movimento de retemporização, o agente A_{ret} emprega as mesmas funções de probabilidade $pPegar()$ e $pDeixar()$ utilizadas pelos agentes A_{lb} e A_{iob}. No entanto, a sua função de atratividade é implementada de forma distinta.

A Equação 3.5 mostra o cálculo da função de atratividade R empregada pelos agentes A_{ret}, no qual $f = R(T)$. Cada agente A_{ret} mantém um histórico local, T_{loc}, de atraso do caminho pelo qual passou, e um histórico global, T_{gl}, que é constantemente atualizado para conter o valor estimado para o menor atraso entre dois registradores no circuito. Quando um agente A_{ret} tem a possibilidade de movimentar um registrador, ele verifica a diferença do seu histórico local em comparação com o global para tomar a decisão. O agente então pega o registrador com probabilidade $pPegar(ag)$, como é descrito no Algoritmo 2. Se o atraso do caminho percorrido for considerado significativo, esse caminho deve ser segmentado. Por esse motivo, o agente muda a direção de seu movimento quando pega um registrador. Uma vez que o agente A_{ret} possui um registrador, ele armazena seu histórico local T_{loc} em um histórico anterior T_{ant} e começa um novo histórico local T_{loc}.

$$f = \begin{cases} 1.0 & \text{se } T_{gl} >= T_{loc} \\ \frac{T_{gl}}{T_{loc}} & \text{se } T_{gl} < T_{loc} \end{cases} \tag{3.5}$$

Visto que a Equação 3.5 considera uma estimativa de menor período atual e o atraso do caminho percorrido pelo agente, isto a torna adequada para o cálculo da função de atratividade para a probabilidade de pegar um registrador, $pPegar$. No entanto, a probabilidade de deixar um registrador, $pDeixar$, exige uma abordagem diferente, pois T_{gl} pode ser muito menor em relação ao atraso do caminho percorrido pelo agente. Logo, o agente deve considerar o atraso anterior, T_{ant}, e o atraso do caminho atual T_{loc} para reposicionar o registrador. O objetivo é deixar esse registrador próximo a metade do caminho anteriormente percorrido para segmentá-lo de forma balanceada. A nova função de atratividade R', definida na Equação 3.6, na qual $f' = R'(T)$, leva em conta T_{ant} e T_{loc}.

$$f' = \frac{2 * T_{loc}}{T_{ant}} \tag{3.6}$$

Os agentes A_{ret} inspecionam diretamente as informações internas aos blocos básicos para efetuar a movimentação dos registradores. O registrador está **habilitado** em um bloco básico se a saída do bloco básico é a saída do registrador. Se, por outro lado, a saída da LUT é a saída do bloco básico, o registrador está **desabilitado**. Desse modo, seja ble_{at} o bloco básico atualmente associado a um agente de retemporização A_{ret}. Se o agente A_{ret} retira o registrador de ble_{at}, ele desabilita o registrador de ble_{at}, ficando **ocupado** até deixá-lo em outro bloco básico. Por outro lado, se A_{ret} posiciona um registrador em ble_{at}, ele habilita o registrador de ble_{at} e fica **livre** para movimentar outros registradores.

Ao contrário dos registradores já presentes no circuito, os novos registradores adicionados pela segmentação não pertencem à rede de interconexões internas. Somente após serem movimentados pelos agentes A_{ret}, eles se tornam parte da rede de interconexões. O Algoritmo 2 descreve como os agentes A_{ret} efetuam essa tarefa de retemporização.

Inicialmente, os agentes são gerados com probabilidade P_{gen} e associados a cada um dos registradores que foram criados nas entradas e saídas do circuito (l.2). Uma vez que um agente A_{ret} chega a um novo bloco básico, se ele estiver ocupado e o bloco básico estiver com o registrador desabilitado, ele pode habilitá-lo e ficar livre para movimentar outros registradores (l.6-11). Caso o agente A_{ret} esteja livre e decida retirar um registrador de um bloco básico cujo registrador está habilitado, ele se torna então ocupado e só ficará livre quando reposicionar esse registrador (l.12-20). A operação de pegar o registrador é seguida de uma mudança de direção no movimento (l.18), para frente ou para trás. Isso é feito porque o agente somente retira o registrador se o atraso percorrido for considerado grande, o que leva a necessidade de segmentar o caminho percorrido para reduzir seu atraso. Ao final do tempo de vida de todos os agentes, o período de *clock* é avaliado para verificar se é satisfatório (l.25).

Algoritmo 2 Processo de retemporização.

```
1:  procedimento RETEMPORIZACAO(Agentes, FPGA)
2:      geraAgentes()        ▷ Gera os agentes A_ret e associa aos registradores previamente adicionados
3:      enquanto Agentes ≠ ∅ faça
4:          para todos ag ∈ Agentes faça
5:              ble_at ← ag.pos
6:              se ag.circuito ≠ LIVRE então
7:                  r ← rand()
8:                  se pDeixar(ag) >= r and ble_i.registrador = DESABILITADO então
9:                      ble_i.registrador ← HABILITADO
10:                     ag.circuito ← LIVRE
11:                 fim se
12:             senão
13:                 ble_i ← escolheBLE()   ▷ Se o registrador do bloco básico atual estiver ativo, o mesmo bloco é escolhido
14:                 se ble_i ≠ NULL então
15:                     se pPegar(ag) ≥ r and ble_i.registrador = HABILITADO então
16:                         ble_i.registrador ← DESABILITADO
17:                         ag.circuito ← OCUPADO
18:                         mudaDirecao(ag)      ▷ Altera a direção do movimento (para saída/entrada) de forma a segmentar o caminho percorrido
19:                     fim se
20:                 fim se
21:             fim se
22:             movAgSin(ag)      ▷ Movimenta o agente, efetuando as operações e correções necessárias nas conexões
23:         fim para
24:     fim enquanto
25:     avaliaPeriodoClock() ▷ Faz uma varredura global verificando se o período de clock é satisfatório
26: fim procedimento
27: procedimento MOVAGSIN(ag)
28:     ble_at ← ag.pos
29:     se ag.circuito = LIVRE então
30:         ag.pos ← escolheBloco(ble_at.LUT.entradas, ble_at.LUT.saidas, direcao(ag))
31:     senão          ▷ Agente está carregando um registrador, portanto o movimento deve respeitar as restrições de retemporização
32:         se direcao(ag) = D_saida então
33:             moveSaida(ag, ble_at)
34:         senão                                                    ▷ direcao(ag) = D_entrada
35:             moveEntrada(ag, ble_at)
36:         fim se
37:     fim se
38: fim procedimento
```

O procedimento $movAgSin(ag)$ (l.22) descreve as operações realizadas pelo agente A_{ret} ao se mover para outra posição. Nessa movimentação, o agente que não possui um registrador escolhe uma das entradas do bloco básico atual ble_{at} quando se move na direção das entradas ou escolhe um dos destinos de ble_{at} quando se move em direção as saídas (l.29-30). Se o agente possui um registrador, no entanto, algumas restrições de movimento devem ser respeitadas, pois envolvem a movimentação de um registrador gerando alterações nas conexões do circuito (l.31-36). Essas restrições dependem da direção do movimento do agente (l.33, l.35).

A movimentação dos registradores em direção às entradas é descrita no Algoritmo 3. Essa movimentação depende de vários fatores, mas é possível sempre que existirem blocos básicos livres para ajustar as interconexões. Caso um agente A_{ret} decida mover o registrador do seu bloco básico atual ble_{at} para trás, esse registrador deve ser movido da saída da LUT de ble_{at} para as conexões das suas entradas ativas (l.2-3).

Um bloco básico conectado a uma entrada i da LUT é chamado de ble_i. Quando o único destino de ble_i é o bloco ble_{at}, uma cópia $agCp_i$ do agente A_{ret} pode se encaminhar diretamente a ble_i com uma cópia do registrador (l.4-7). Caso contrário, ble_{at} não é o único destino de ble_i e os demais destinos devem ser isolados desse movimento de retemporização. Para isso, o agente A_{ret} aloca um bloco básico intermediário entre ble_i e ble_{at}. Através da função $encontraBLEVazio()$, o agente tenta alocar esse bloco, chamado de **bloco registrador**, o mais próximo possível de ble_i (l.9-12). O agente habilita o registrador desse bloco antecipando a operação seguinte ("deixar") e direciona suas cópias $agCp_i$ diretamente aos blocos ble_i, livres para movimentarem outros registradores (l.13-14).

Os blocos registradores adicionados nas saídas de ble_i futuramente podem ser substituídos por um único registrador, evitando o desperdício de recursos. O agente A_{ret} pode inserir esse registrador único ou no bloco básico ble_i, cuja saída se conecta aos blocos registradores, ou em um único bloco registrador entre ble_i e esses blocos registradores. Para efetuar essa modificação, os agentes sempre verificam após sua operação normal, se é possível liberar os blocos registradores em excesso gerados por movimentações anteriores. Essa tarefa é efetuada pelo procedimento $ajustaConexao()$ (l.19), cujas operações são descritas em l.23-45.

A movimentação dos registradores em direção às saídas é descrita em detalhes no Algoritmo 4. Um agente A_{ret} só move os registradores para a saída de um bloco básico ble_{at} se todas as entradas ativas dos blocos básicos de destino, $ble_d.LUT.entradas(j)$, estão conectadas com blocos básicos de origem ble_j que tenham registradores habilitados (l.5-15). Neste caso, o agente remove todos os registradores presentes nas entradas do bloco básico ble_{at} e adiciona um registrador a sua saída.

Quando os blocos básicos ble_j tiverem mais de um destino além de ble_d, deve-se manter os registrador desses destinos. Para isso, os agentes criam blocos registradores nessas conexões (l.7-11). Os agentes agendam tais modificações através das funções $adicionaBLE$ e $adicionaConexao()$ e as efetuam usando a função $atualizaRegs()$ (l.23). Da mesma forma, os agentes mantêm os registradores dos blocos ble_d que tiverem o registrador habilitado e inserem um bloco registrador entre a saída de ble_d e seus destinos, $ble_d.netDests$, para que haja dois registradores nessa conexão (l.19-21).

Algoritmo 3 Movimentação do agente de retemporização em direção das entradas.

```
1: procedimento MOVEENTRADA(ag, ble_at)
2:     para i = 1 até ble_at.LUT.entradas.tam faça
3:         se ble_at.LUT.entradas(i) = ATIVA então
4:             ble_i ← ble_at.LUT.entradas(i)
5:             agCp ← geraAgente(ag)
6:             se ble_i.netDests.tam = 1 então
7:                 agCp.circuito ← OCUPADO
8:             senão
9:                 ble_n ← encontraBLEVazio(ble_i.blocoLogico)        ▷ Escolhe a partir do bloco lógico ble_i.blocoLogico um bloco básico não utilizado
10:                ble_n.blocoRegistrador ← TRUE
11:                ble_n.conexaoAdd(ble_i, ble_d)
12:                insereInterconexao(ble_i, ble_n, ble_d)
13:                ble_n ← HABILITADO
14:                agCp.circuito ← LIVRE        ▷ Como o registrador já foi habilitado em ble_n, a cópia agCp segue livre para ble_i
15:            fim se
16:            agCp.pos ← ble_i
17:        fim se
18:    fim para
19:    ajustaConexao(ag, ble_at)        ▷ Otimiza recursos. Blocos registradores gerados em movimentos anteriores podem ser fundidos em um só
20:    removeAgente(ag)
21: fim procedimento
22: procedimento AJUSTACONEXAO(ag, ble_at)
23:    hab ← TRUE
24:    para i = 1 até ble_at.netDests.tam faça        ▷ Verifica se os blocos registradores nas conexões de saída de ble_at estão habilitados, eles podem ser fundidos em um só registrador
25:        ble_d ← ble_at.netDests(i)
26:        se ble_d.blocoRegistrador ≠ TRUE ou ble_d.registrador ≠ HABILITADO então
27:            hab ← FALSE
28:        fim se
29:    fim para
30:    se hab ≠ FALSE então
31:        se ble_at.registrador = HABILITADO então        ▷ Se ble_at está habilitado, o novo bloco registrador ble_n é criado em sua saída
32:            ble_n ← encontraBLEVazio(ble_at.blocoLogico)
33:            ble_n.blocoRegistrador ← TRUE
34:            ble_n.conexaoAdd(ble_at, ble_at.netDests)        ▷ Bloco ble_n é intermediário entre ble_at e seus blocos básicos de destino ble_at.netDests
35:            ble_n ← HABILITADO
36:            insereInterconexao(ble_at, ble_n, ble_at.netDests)
37:        senão        ▷ Caso contrário ble_at é habilitado para conter o novo registrador
38:            ble_at.registrador ← HABILITADO
39:        fim se
40:        para i = 1 até ble_at.netDests.tam faça
41:            ble_d ← ble_n.netDests(i)
42:            ble_n.netDests(i) ← ble_d.netDests(1)        ▷ Restaura conexão anterior
43:            liberaBLE(ble_d)
44:        fim para
45:    fim se
46: fim procedimento
```

Algoritmo 4 Movimentação em direção às saídas.

```
1: procedimento MOVESAIDA(ag, ble_at)
2:     para i = 1 até ble_at.netDests.tam faça
3:         ble_d ← ble_at.netDests(i)
4:         para j = 1 até ble_d.LUT.tam faça        ▷ Verifica se todas as entradas ativas da LUT contém registradores habilitados
5:             se ble_d.LUT.entradas(j) = ATIVA então
6:                 ble_j ← ble_d.LUT.entradas(j)
7:                 se ble_j.registrador = HABILITADO and ble_j ≠ ble_at então
8:                     se ble_j ∉ listaRegs então
9:                         adicionaBLE(listaRegs, ble_j)    ▷ Os blocos básicos que tiveram suas conexões alteradas são colocados em uma lista
10:                    fim se
11:                    adicionaConexao(listaCon, ble_j, ble_d)    ▷ O bloco registrador que será criado irá conectar ble_j a ble_d
12:                senão
13:                    return RETEMPORIZACAO_INVALIDA
14:                fim se
15:            fim se
16:        fim para
17:        ag_d ← geraAgente(ag)
18:        ag_d.pos ← ble_d
19:        se ble_d.registrador = HABILITADO então
20:            insereBlocoReg(ble_d, DEST)    ▷ Insere um bloco registrador entre ble_d e suas interconexões ble_d.netDests
21:        fim se
22:    fim para
23:    atualizaRegs(listaRegs, listaCon)    ▷ As conexões dos blocos ble_j presentes em listaRegs e não em listaCon, recebem um bloco registrador intermediário
24: fim procedimento
```

3.4.1 Exemplos de retemporização

A Figura 3.5 ilustra o processo de retemporização no qual um agente A_{ret} efetua a movimentação do registrador em direção às entradas (movimento de retemporização para trás). A figura da esquerda mostra o estado do circuito antes da retemporização, no qual o circuito contém quatro blocos lógicos com dois blocos básicos cada, estando um desses blocos lógicos, bl_{at}, ampliado em destaque. Já a figura da direita ilustra o estado do mesmo circuito após a retemporização.

No processo de retemporização, através do Algoritmo 2, um agente A_{ret}, que se move em direção às saídas, chega a bl_{at} e escolhe o bloco ble_0 como seu bloco básico atual (Fig-Esquerda). O bloco básico ble_0 contém um registrador habilitado e o agente decide pegá-lo após verificar a probabilidade de pegar $pPegar(ag)$. Uma vez que o agente A_{ret} pega o registrador em ble_0, ele muda a direção de seu movimento, passando a se movimentar em direção às entradas.

Para se movimentar em direção às entradas, o agente A_{ret} se utiliza do Algoritmo 3. Dessa forma, o agente verifica as entradas ativas da LUT de ble_0, isto é, os blocos básicos b_0, b_1 e b_2, para decidir como o movimento pode ser efetuado. Como esses blocos básicos não contêm outro destino exceto ble_0 e seus registradores estão desabilitados, o registrador

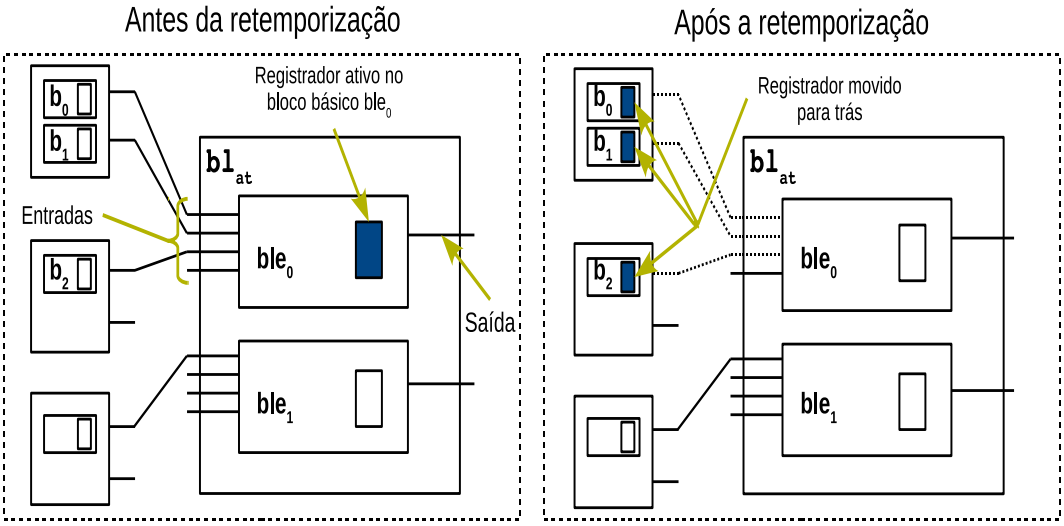


Figura 3.5: Movimento de um registrador para trás no processo de retemporização.

de ble_0 pode ser movido diretamente para esses blocos básicos. Assim, o agente A_{ret} cria três copias de si mesmo, chamadas $agCp_i | i \in \{0, 1, 2\}$, sendo que a cada $agCp_i$ é associado um registrador. Esses agentes se encaminham até os blocos básicos b_0, b_1 e b_2, nos quais eles posicionam seus registradores. Ao final dessa operação, o registrador em ble_0 é desabilitado (removido) e três novos registradores foram criados (Fig-Direita).

A Figura 3.6 ilustra o processo de retemporização no qual um agente A_{ret} efetua a movimentação do registrador em direção às saídas (movimento de retemporização para frente). A figura da esquerda mostra o estado do circuito antes da retemporização, no qual o circuito contém três blocos lógicos com dois blocos básicos cada[1], estando dois desses blocos lógicos, bl_{or} e bl_d, ampliados em destaque. A figura da direita ilustra o estado do mesmo circuito após a retemporização.

No processo de retemporização, através do Algoritmo 2, um agente A_{ret}, que se move em direção às entradas, chega a bl_{or} e escolhe b_2 como seu bloco básico atual (Fig-Esquerda). O bloco básico b_2 contém um registrador habilitado e o agente decide pegá-lo após verificar a probabilidade de pegar $pPegar(ag)$. Uma vez que o agente A_{ret} pega o registrador em b_2, ele muda a direção de seu movimento, passando a se movimentar em direção às saídas.

Para se movimentar em direção às saídas, o agente A_{ret} se utiliza do Algoritmo 4. Dessa forma, o agente verifica os destinos da saída de b_2 para decidir como o movimento pode ser efetuado. A saída do bloco básico b_2 possui somente um destino, o bloco básico ble_0, presente no bloco lógico bl_d. O agente então averigua se as outras entradas ativas de ble_0 também contém um registrador habilitado. Caso alguma dessas entradas ativas não possua o registrador habilitado, o movimento em direção à saída não é possível. Como neste exemplo b_0 e b_1 contém um registrador habilitado e suas saídas não possuem outro

[1]Somente são mostrados na figura os blocos básicos envolvidos no movimento de retemporização

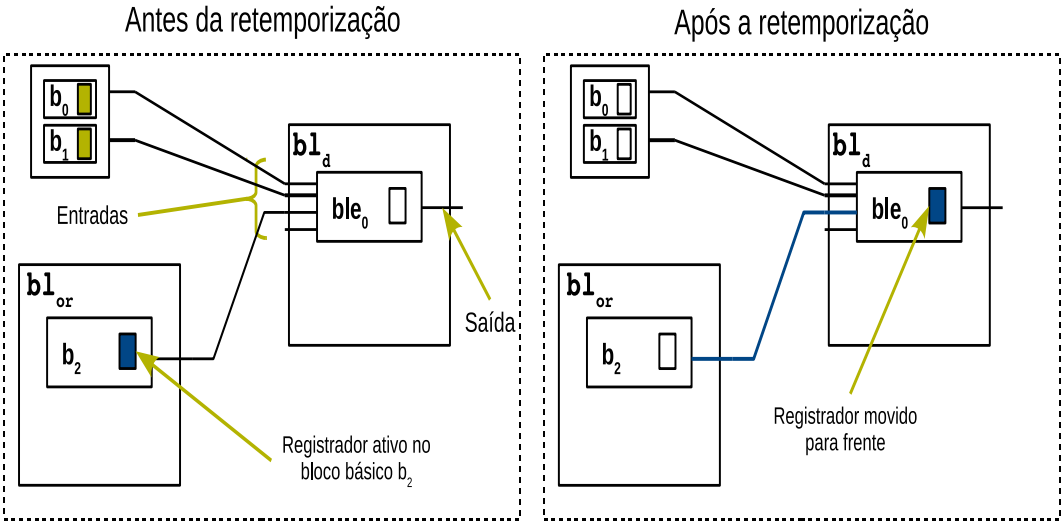


Figura 3.6: Movimento de um registrador para frente no processo de retemporização.

destino exceto ble_0, o movimento é válido e o agente desabilita (remove) o registrador dos blocos básicos b_0 e b_1. Em seguida ele se encaminha para o bloco básico ble_0 no qual posiciona o seu registrador (Fig-Direita).

3.5 Conclusão

Este capítulo apresentou a técnica bio-inspirada **AntES** para economia de energia em FPGAs. Esta, é composta pelas etapas de segmentação, seguida de posicionamento e retemporização realizados de forma simultânea. Essa realização simultânea de posicionamento e retemporização é facilitada pela natureza distribuída do algoritmo bio-inspirado proposto. Agentes inspirados em formigas posicionam os blocos lógicos e blocos de I/O ao passo que outros agentes realizam a movimentação de registradores.

Uma especificação da técnica **AntES** foi apresentada através de equações e algoritmos que descrevem os detalhes da solução. Essa técnica tem como base uma especificação, consistindo de equações que definem o comportamento de cada agente e algoritmos que descrevem os detalhes das suas operações. Exemplos mostram o processo de posicionamento de duas portas lógicas por dois agentes de posicionamento, podendo-se observar como operações simples dos agentes resultam em um bom posicionamento dessas portas lógicas. O movimento de retemporização é ilustrado através de um exemplo, no qual um agente movimenta um registrador em direção às entradas e outro exemplo no qual ele o move em direção às saídas. Os agentes de posicionamento e de retemporização, embora sejam apresentados em contextos distintos, trabalham em paralelo.

CAPÍTULO 4

AVALIAÇÃO DE DESEMPENHO DA TÉCNICA DE POSICIONAMENTO BIO-INSPIRADA

Este capítulo apresenta a avaliação do desempenho da técnica de posicionamento bio-inspirada desenvolvida. A Seção 4.1 apresenta o ambiente de desenvolvimento utilizado para implementar a solução. Os parâmetros importantes e os *benchmarks* usados na avaliação são detalhados na Secão 4.2. A Seção 4.3 define as métricas e a maneira que elas são usadas na avaliação. Por fim, a Seção 4.4 apresenta os resultados obtidos comparando o desempenho da técnica **AntES** com a versão padrão do algoritmo implementado no VPR 5.0.

4.1 Ambiente de desenvolvimento

Atualmente, muitas empresas especializadas em FPGAs desenvolvem ferramentas CAD. Estas ferramentas contêm o fluxo de projeto completo desde a edição esquemática e em linguagem de descrição de *hardware* à programação do FPGA. Dentre elas, destacam-se o QuartusII da Altera [52], ISE [53] da Xilinx e Libero da Actel [54]. Elas desenvolvem o *software* de edição em alto nível e a parte final do fluxo CAD que depende do *hardware* de destino, utilizando-se de outras ferramentas na parte intermediária do fluxo, como o simulador de circuitos ModelSim [55] desenvolvido pela Mentor Graphics. Visto que essas ferramentas são proprietárias e seu código é fechado, a implementação de novas abordagens para o fluxo de projeto utilizando diretamente o código proprietário não é possível [1].

A ferramenta CAD de código aberto mais utilizada no meio acadêmico para posicionamento e roteamento em FPGAs é o *Versatile Place and Route (VPR)*. O VPR geralmente é usado como indicador da qualidade de novas soluções para posicionamento e roteamento por obter resultados eficientes. Além disso, o VPR é o núcleo da ferramenta de posicionamento e roteamento do Quartus II [25]. Outras soluções desenvolvidas baseadas no VPR também já foram adotadas no mercado, como o posicionamento ciente do *clock* [38], adotado pela Xilinx como parte do ISE [10].

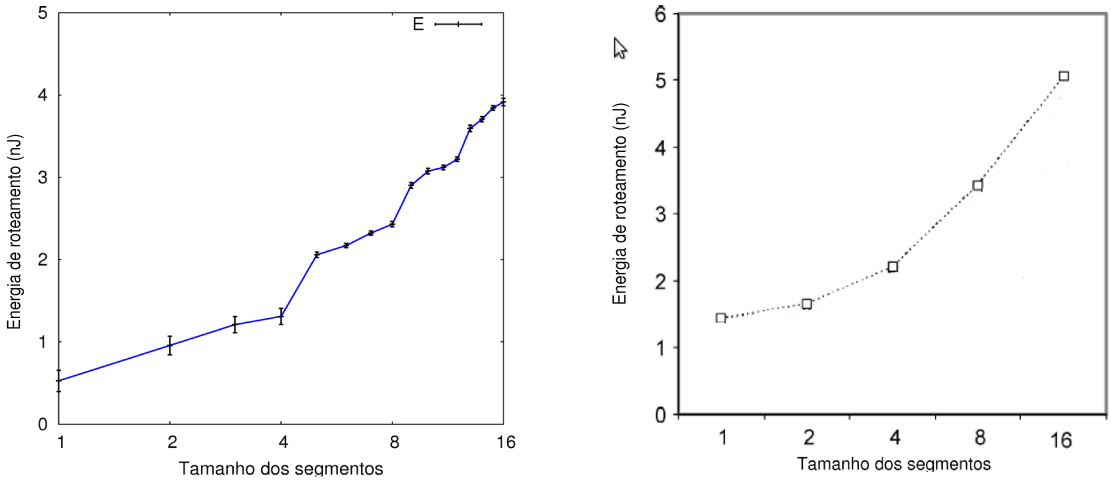
A maioria dos trabalhos relacionados que utilizaram o VPR em suas pesquisas se basearam em suas versões anteriores. No entanto, Jamieson et al. acrescentaram novas características ao VPR, que estão disponíveis na versão 5.0 [56]. Nessa versão, eles implementam o modelo de energia proposto por Poon et al. [12]. Por esse motivo, o VPR

[1] A Altera oferece a possibilidade de adição de novas ferramentas em diferentes pontos do fluxo CAD através do **QUIP**

5.0 é utilizado neste trabalho.

4.1.1 Validação do ambiente de desenvolvimento e simulação

Para validar o ambiente, um experimento feito por Poon et al. em [12] utilizando a versão 4.3 do VPR foi reproduzido na versão 5.0, comparando os seus resultados. Para tal foi utilizado o mesmo cenário de experimentação de [12]. Neste cenário, as LUTs têm quatro entradas e cada bloco lógico tem quatro blocos básicos. O comprimento do segmento é variado de 1 a 16. Diferentes curvas foram mostradas no gráfico original, para diferentes topologias de caixa comutadora. Por simplicidade, neste trabalho foi escolhida a curva da topologia "wilton" para a comparação.



(a) VPR versão 5.0 (b) VPR versão 4.3

Figura 4.1: Energia dissipada no circuito de roteamento versus tamanho dos segmentos.

No entanto, parâmetros importantes como a população de caixas de conexão e caixas comutadoras por trilha não foram descritos e tiveram que ser geradas aleatoriamente. O experimento usa os 20 maiores *benchmarks* MCNC [2] [57] como entrada para o VPR. A média aritmética do consumo de energia do circuito de roteamento desses *benchmarks* é calculada, gerando um valor único. Para evitar que casos especiais comprometessem o resultado, esse procedimento foi efetuado 35 vezes, com um intervalo de confiança de 95%.

A Figura 4.1(a) mostra a variação no tamanho dos segmentos no eixo x e a energia dissipada no eixo y (em *nanoJoule*) para a versão 5.0 do VPR. Pode-se notar diferenças nos valores absolutos da energia dissipada quando comparados aos da versão 4.3 mostrada na Figura 4.1(b). No entanto, o comportamento apresentado pela curva é semelhante. Portanto, essa versão do VPR pode ser utilizada na estimativa do consumo de energia e na validação da solução deste trabalho.

[2] Microelectronics Center of North Carolina (MCNC) benchmark suite.

4.2 Cenário de avaliação

O posicionamento de circuitos em um FPGA utiliza informações sobre a tecnologia de destino. Essas informações da arquitetura do FPGA são definidas em um arquivo XML passado como parâmetro inicial do VPR. Nesse arquivo estão informações importantes, como o número de blocos básicos por bloco lógico e a topologia das caixas comutadoras. A Tabela 4.1 resume os parâmetros que descrevem a arquitetura do FPGA considerado na avaliação.

Tabela 4.1: Parâmetros da arquitetura do FPGA.

Parâmetros	**Opção utilizada**
Número de LUTs por bloco lógico	4
Número de entradas da LUTs	4
Capacidade do banco de I/O (em blocos de I/O)	8
Tamanho do segmentos (em espaço de blocos lógicos)	1
Topologia da caixa comutadora	wilton
% de segmentos de roteamento conectados às entradas	15%
% de segmentos de roteamento conectados às saídas	10%
Entradas por bloco lógico	10
Tamanho do FPGA	auto

Como se pode ver pela tabela, o tamanho do FPGA é definido em tempo de execução (*auto*), portanto ele depende do tamanho do circuito *benchmark* que será posicionado. Os parâmetros importantes do VPR são mostrados na Tabela 4.2. Alguns deles são específicos de cada alternativa de posicionamento, neste caso a alternativa que não possui o parâmetro recebe como valor o caractere '-'.

Tabela 4.2: Parâmetros dos algoritmos de posicionamento.

Parâmetros	**VPR**	**AntES**
CPD X BB (λ)	0.5	0.5
PW X BB (σ)	-	1.0
Probabilidade de gerar um agente (P_{gen})	-	0.8
Constante da operação *pegar* (k_p)	-	0.7
Constante da operação *deixar* (k_d)	-	0.5

Para comparar os resultados da solução **AntES** com o VPR, são utilizados diferentes circuitos do conjunto de *bechmarks* MCNC. Eles são classificados, no cenário de avaliação usado neste trabalho, nas categorias de *benchmark*: **pequeno**, **médio** ou **grande**. Os circuitos são passados para o VPR, respeitando o seu formato de interconexões. Além do arquivo para mapeamento de interconexões, são utilizados também um arquivo de função e um arquivo de estimativa de atividade gerado pelo ACE [3]. O arquivo de função contém

[3]Activity Estimation tool for FPGAs.

as tabelas verdade pra cada LUT, enquanto o arquivo de estimativa de atividade contém as densidades e probabilidades de transição de cada sinal.

Por simplicidade, um subconjunto dos circuitos *benchmarks* MCNC foi selecionado neste trabalho para comparar os algoritmos de posicionamento em um cenário genérico. Esses *benchmarks* são mostrados na Tabela 4.3, que descreve para cada um deles sua quantidade de portas lógicas, o tipo do circuito e a sua categoria. Como se pode observar, foram escolhidos tanto circuitos sequenciais quanto combinacionais em cada categoria.

Tabela 4.3: *Benchmarks* usados para a avaliação de desempenho.

Benchmark	**Portas lógicas**	**Tipo**	**Categoria**
cm42a	17	combinacional	pequeno
cm138a	17	combinacional	pequeno
s27	10	sequencial	pequeno
i1	46	combinacional	pequeno
f51m	43	combinacional	pequeno
s208.1	104	sequencial	médio
cordic	102	combinacional	médio
frg1	105	combinacional	médio
s298	119	sequencial	médio
s820	289	sequencial	médio
alu4	681	combinacional	grande
i10	2260	combinacional	grande
dsip	2097	sequencial	grande

Para avaliar a eficiência do posicionamento para circuitos com potencial utilização em redes corporais, alguns *benchmarks* MCNC foram selecionados de forma a representar um cenário de redes corporais. Este cenário considera uma rede corporal que possui um nó agregador (*clusterhead*). Este *clusterhead* contém um FPGA responsável por seu processamento de dados e o controle de sua atividade de sensoriamento. Esses *benchmarks* são mostrados na Tabela 4.4, que também mostra o tamanho e o tipo do circuito. Esses *benchmarks* foram selecionados considerando os requisitos de privacidade e segurança, processamento de dados e comunicação, importantes para adoção dessas redes em larga escala [2]. Os circuitos mult32a, alu4, count e C1355 representam operações básicas de processamento, enquanto o dsip implementa criptografia e o clma representa a comunicação através de um controlador de barramento.

4.3 Métricas

A avaliação do desempenho, utilizando os cenários descritos anteriormente, é feita através da comparação dos resultados da **AntES** com os resultados obtidos pelo VPR nas seguintes métricas: **atraso do caminho crítico**, **potência dissipada**, **comprimento total**

Tabela 4.4: *Benchmarks* representativos de BANs.

Benchmark	**Portas lógicas**	**Tipo**	**Categoria**
count	143	combinacional	médio
mult32a	416	sequencial	médio
C1355	546	combinacional	grande
s1488	653	sequencial	grande
alu4	681	combinacional	grande
dsip	2097	sequencial	grande
clma	35000	sequencial	grande

das conexões e **período de clock mínimo**. A última é aplicada na avaliação da retemporização, enquanto as demais são usadas na avaliação final após o posicionamento e o roteamento.

Inicialmente, as métricas são calculadas considerando apenas o posicionamento sem a retemporização. O **atraso do caminho crítico** é observado para verificar se a velocidade de operação do circuito no FPGA não é comprometida pela nova técnica de posicionamento. Posteriormente, a retemporização ameniza o atraso do caminho crítico, segmentando esse caminho.

O **período de clock mínimo** é confrontado com o período de *clock* que o circuito teria se ele fosse perfeitamente segmentado. Seja A o conjunto dos agentes de retemporização e d_{ag} o atraso percorrido pelo agente ag, a Equação 4.1 mostra o cálculo de uma estimativa instantânea de período, C_{del}. Essa estimativa é comparada ao período de *clock* mínimo, P_{min}, calculado através da Equação 4.2. Nessa equação, P_{at} é o período atual e S é número de segmentos em que o circuito deve ser dividido. Uma relação entre o resultado real (C_{del}) e o ideal (P_{min}) será analisada, de forma a determinar se o período de *clock* obtido é satisfatório. Dessa forma, a etapa de retemporização será avaliada em termos de velocidade, pois embora o foco seja a redução no consumo de energia, a retemporização deve reduzir o período de *clock* permitindo o circuito operar de forma mais rápida.

$$C_{del} = \max \left\{ p \in \bigcup_{ag \in A}^{d_{ag}} \right\} \tag{4.1}$$

$$P_{min} = \frac{P_{at}}{S} \tag{4.2}$$

A **potência dissipada** é avaliada inicialmente para o posicionamento de forma isolada e posteriormente para o posicionamento e segmentação de forma simultânea. O foco é otimizar a potência dinâmica, principal fonte do consumo de energia nos FPGAs. O processo de posicionamento que leva em conta a potência dinâmica tem como base a estimativa de atividade para cada interconexão do circuito, buscando encurtar a distância entre as interconexões mais ativas.

As interconexões (*nets*) têm como origem um bloco lógico e como destino um ou mais blocos lógicos. A caixa delimitadora de uma interconexão é o menor retângulo que contém todos os blocos ligados a essa interconexão. O **comprimento total das conexões** é estimado somando as caixas delimitadoras de todas as interconexões [27].

O impacto do número de agentes nos resultados também será observado em busca de um valor aproximado para o número ótimo de agentes de cada tipo. Será observada a fração de agentes A_{lb} e A_{iob} em relação ao número de blocos lógicos e blocos de I/O que obtém a melhor redução no consumo de energia. Do mesmo modo, a fração de agentes A_{ret} em relação ao número de registradores adicionados pela segmentação (*C-Slow*) que obtém os melhores resultados em termos de potência dissipada e período de *clock* mínimo.

4.4 Avaliação do desempenho

Esta seção apresenta a avaliação do desempenho da solução **AntES**. Primeiro, os parâmetros importantes nas operações dos agentes são definidos através da análise do impacto causado nas métricas. Definidos esses valores, o posicionamento **AntES** é avaliado, comparando o seu desempenho ao obtido pelo VPR.

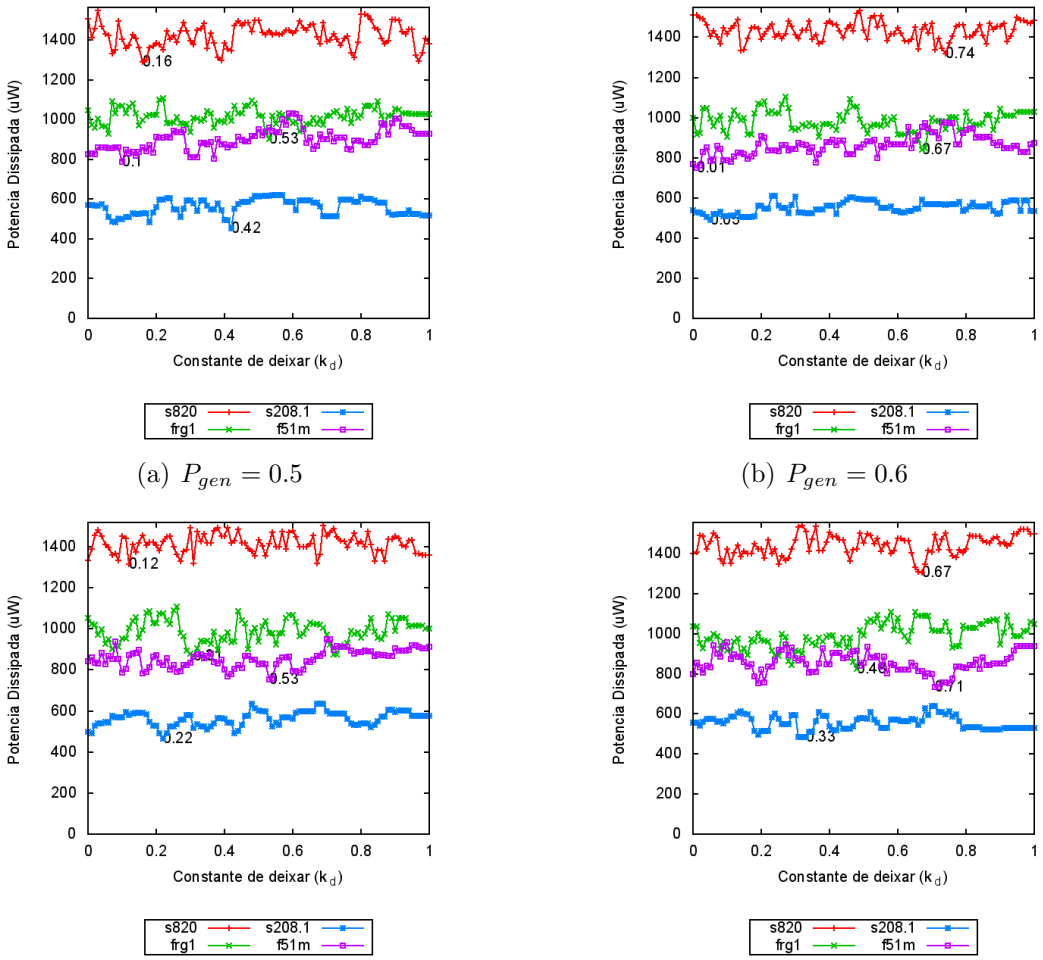
4.4.1 Operações dos agentes

Cada posição no FPGA que possa receber um bloco lógico ou bloco de I/O tem uma probabilidade P_{gen} de gerar um agente. Essa probabilidade é ajustada no código da **AntES** de forma a gerar quantidades semelhantes de agentes A_{iob} e A_{lb}. Uma vez gerados os agentes, suas operações durante a execução do algoritmo se baseiam nas funções de probabilidade $pPegar$ e $pDeixar$ mostradas nas Equações 3.1 e 3.2 discutidas no capítulo 3. Essas funções auxiliam na decisão do agente de pegar ou deixar um bloco em uma determinada posição. Elas são compostas, além da função de atratividade, pelas constantes de pegar e deixar, k_p e k_d, cujos valores servem para facilitar ou dificultar as operações de pegar ou deixar.

Nos experimentos a seguir, as contantes k_p e k_d são variadas de 0 a 1, para diferentes valores de P_{gen}, a fim de determinar os melhores valores para cada parâmetro. Para cada combinação de valores das constantes e P_{gen} se executa o posicionamento e o roteamento, gerando as estimativas para as métricas ao final dessas execuções. Devido a natureza intensiva desse experimento, por simplicidade, foram escolhidos os *benchmarks* f51m, s208.1, frg1 e s820, sendo o primeiro um *benchmark* pequeno e os demais médios. Para visualizar a influência nos tipos de *benchmark*, dois deles são combinacionais e dois sequenciais.

A potência dissipada com a variação de k_d

A Figura 4.2 mostra a variação da potência dissipada pelo FPGA quando o posicionamento é executado com a constante k_d variando de 0 a 1.0. Para mostrar a influência conjunta dos parâmetros nas métricas, a figura contém gráficos para $P_{gen} = 0.5$, 0.6, 0.7 e 0.8. Nesses experimentos, a constante constante de pegar, k_p, foi fixada em $k_p = 0.1$. Assim, a função de atratividade é o determinante da probabilidade de pegar.



(a) $P_{gen} = 0.5$ (b) $P_{gen} = 0.6$

(c) $P_{gen} = 0.7$ (d) $P_{gen} = 0.8$

Figura 4.2: Potência dissipada com a variação da constante k_d.

O desvio padrão entre os valores de potência dissipada em geral diminui para o *benchmark* pequeno f51m conforme se aumenta P_{gen}, enquanto para os demais *benchmarks* médios o aumento de P_{gen} causa variações maiores, resultando em gráficos menos estáveis em dissipação de potência. Foram coletadas estatísticas para cada curva, como média e desvio padrão dentre outras. Por simplicidade, as estatísticas completas são mostradas apenas para o *benchmark* s820, que por ser sequencial e o maior deste subconjunto é o mais representativo.

A Tabela 4.5 mostra dados estatísticos da execução do *benchmark* médio s820. Os

valores máximos e mínimos são representados como coordenadas (K_d, potência dissipada), enquanto média e desvio padrão em valores de potência dissipada (em μW). Pode-se notar também, que o valor mínimo na potência dissipada para esse *benchmark* foi obtido com $P_{gen} = 0.8$, embora a melhor média tenha sido com $P_{gen} = 0.7$. Isso significa que quando aproximadamente 80% ou 70% das posições do FPGA possuem inicialmente um agente obtém-se os melhores resultados.

Tabela 4.5: Dispersão dos valores de potência dissipada para o *benchmark* s820.

P_{gen}	**Máximo**	**Mínimo**	**Média**	**Desvio padrão**
0.5	(0.16, 1530.82)	(0.04, 1288.08)	1428.787505	47.66962361
0.6	(0.94, 1531.054)	(0.75, 1315.868)	1434.585901	48.97589172
0.7	(0.20, 1530)	(0.24, 1278.976)	1414.581663	54.12880946
0.8	(0.43, 1557.51)	(0.54, 1275.894)	1439.063105	53.57100787

O atraso do caminho crítico com a variação de k_d

O atraso do caminho crítico é o principal determinante da velocidade de processamento do FPGA. Uma solução, mesmo que esteja focada em otimizar outras métricas deve ainda manter uma velocidade de processamento razoável para ser relevante. A variação do atraso do caminho crítico para diferentes valores de k_d e P_{gen} é mostrada na Figura 4.3.

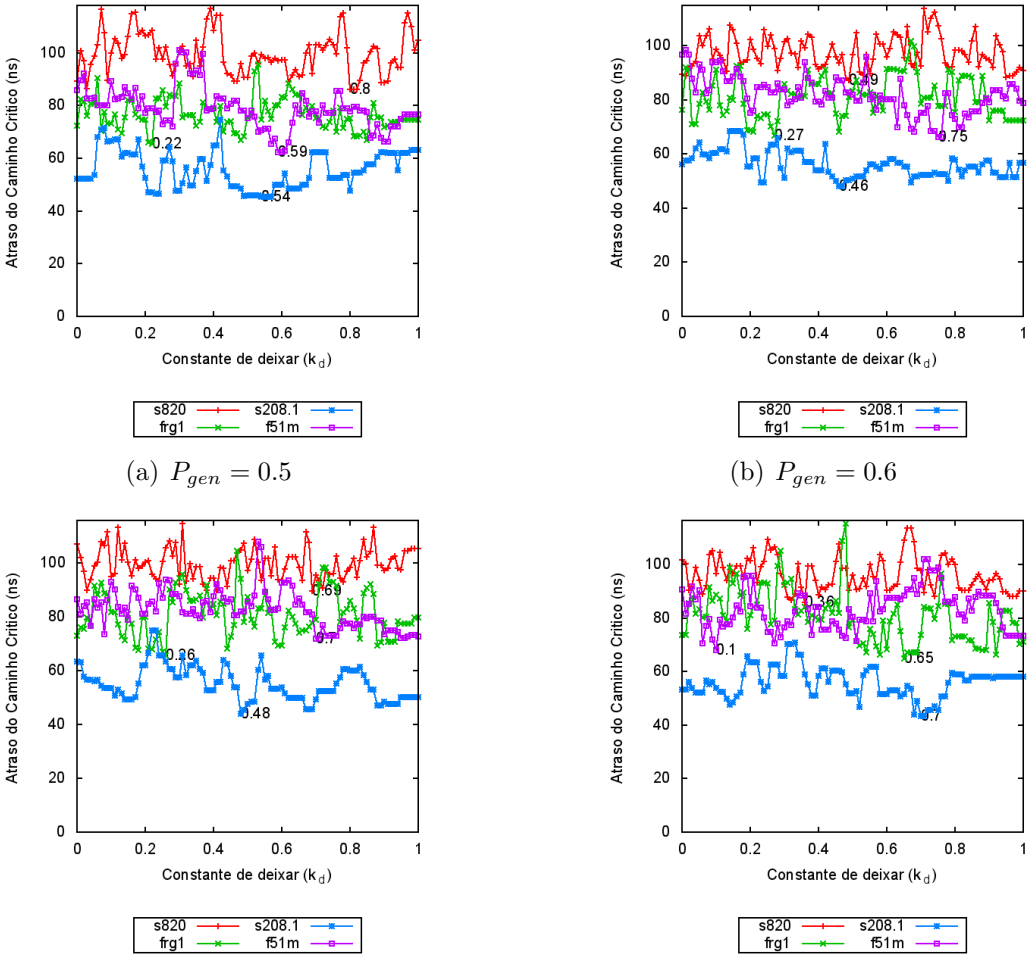
O desvio padrão nos gráficos de atraso do caminho crítico segue um comportamento semelhante ao obtido para a potência dissipada. Os dados estatísticos para o atraso do caminho crítico são mostrados na Tabela 4.6. Os valores do atraso do caminho crítico estão em nanosegundos. Nesse caso, tanto o menor valor mínimo quanto o menor valor médio foram obtidos com $P_{gen} = 0.8$.

Tabela 4.6: Dispersão dos valores de atraso do caminho crítico para o *benchmark* s820.

P_{gen}	**Máximo**	**Mínimo**	**Média**	**Desvio padrão**
0.5	(0.04, 120.9614)	(0.5, 86.00948)	97.74400337	6.39574418
0.6	(0.75, 116.01002)	(0.34, 86.67978)	97.27943822	6.179532189
0.7	(0.24, 121.3365)	(0.2, 85.66296)	99.32129386	6.830499625
0.8	(0.54, 122.43006)	(0.43, 84.71066)	97.04638554	7.711010557

A caixa delimitadora com a variação de k_d

A caixa delimitadora representa o quanto um determinado bloco está próximo dos blocos aos quais está conectado. Por este motivo, esta métrica está bastante ligada à potência dissipada, pois uma caixa delimitadora menor para blocos cujas interconexões são mais ativas leva a uma menor dissipação de energia. A Figura 4.4 mostra a influência de P_{gen} e k_d no tamanho da caixa delimitadora.



(a) $P_{gen} = 0.5$ (b) $P_{gen} = 0.6$

(c) $P_{gen} = 0.7$ (d) $P_{gen} = 0.8$

Figura 4.3: O atraso do caminho crítico com a variação da constante k_d.

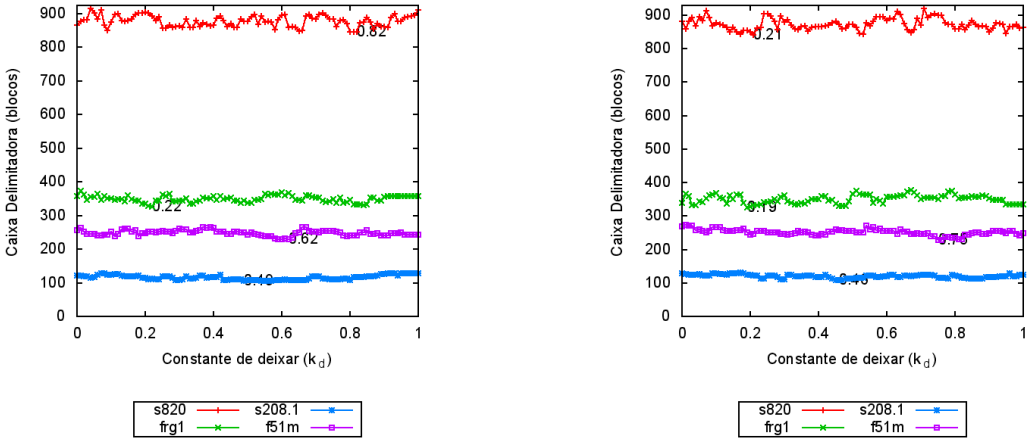
A Tabela 4.7 mostra a dispersão dos valores para o tamanho da caixa delimitadora com a variação de k_d. A menor média foi obtida com $P_{gen} = 0.8$ enquanto o menor valor mínimo foi obtido com $P_{gen} = 0.6$. A partir da análise dos valores mínimos e médios obtidos, foram definidos os valores $P_{gen} = 0.8$ e $k_d = 0.5$ para os demais experimentos.

Tabela 4.7: Dispersão dos valores de atraso do caminho crítico para o *benchmark* s820.

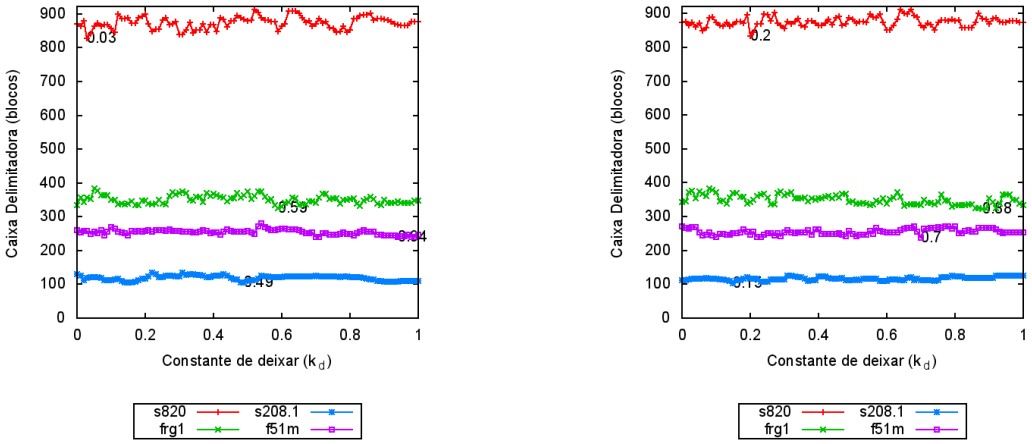
P_{gen}	**Máximo**	**Mínimo**	**Média**	**Desvio padrão**
0.5	(0.30, 906.8)	(0.50, 818)	864.7623762	18.00274897
0.6	(0.19, 893)	(0.08, 816.2)	860.8158416	14.85222361
0.7	(0.23, 907)	(0.02, 822.4)	868.3960396	16.89359595
0.8	(0.61, 913.8)	(0.40, 823.6)	860.7207921	20.53220064

As métricas com a variação de k_p

A Figura 4.5 mostra a influência da constante de pegar k_p na potência dissipada, no atraso do caminho crítico e na caixa delimitadora. Quanto maior o valor de k_t, mais fácil um



(a) $P_{gen} = 0.5$ (b) $P_{gen} = 0.6$



(c) $P_{gen} = 0.7$ (d) $P_{gen} = 0.8$

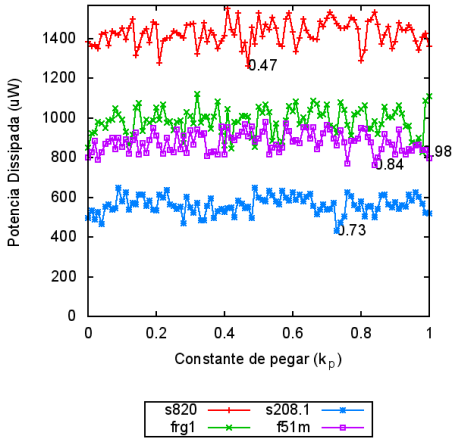
Figura 4.4: A caixa delimitadora com a variação da constante k_d.

agente decidir retirar um bloco de uma determinada posição. Portanto valores mais altos para essa constante leva a um maior número de operações dos agentes, com o potencial de elevar o número de trocas de posições entre blocos.

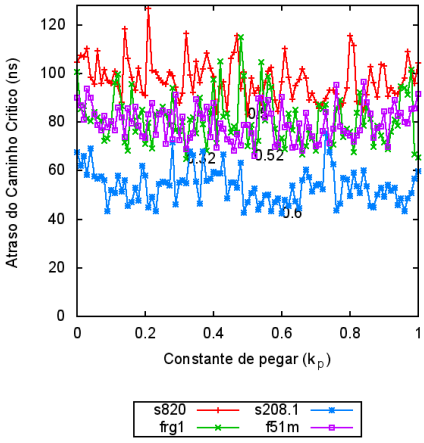
Tabela 4.8: Dispersão dos valores das métricas para o *benchmark* s820.

P_{gen}	**Máximo**	**Mínimo**	**Média**	**Desvio padrão**
PWD	(0.55, 1565.626)	(0.26, 1276.764)	1426.89196	47.38830012
CPD	(0.26, 114.68074)	(0.55, 84.18988)	97.61334059	6.055022278
BB	(0.00, 948.6)	(0.32, 815.2)	850.6574257	22.71634367

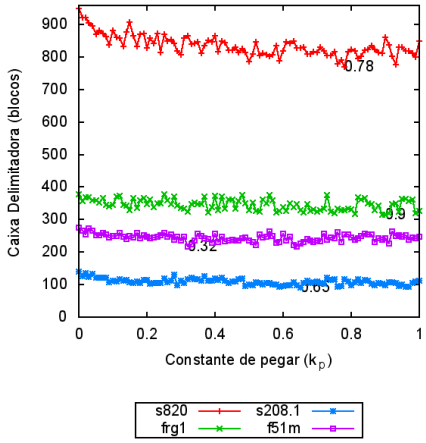
A Tabela 4.8 contém dados de dispersão dos valores de potência dissipada (PWD), atraso do caminho crítico (CPD) e caixa delimitadora (BB) para o *benchmark* s820. Dentre os quatro *benchmarks* utilizados nos experimentos esse é o maior além de ter componentes sequenciais, portanto considerado o mais representativo dentre eles. Pode-se ver, tanto pela Figura 4.5 quanto pela Tabela 4.8, que o impacto da variação de k_p é diferente em cada uma das métricas. A análise gráfica sugere que conforme k_p aumenta, a tendência



(a) Potência dissipada



(b) Atraso do caminho crítico



(c) Caixa delimitadora

Figura 4.5: A influência da variação da constante de pegar (k_p) posicionamento.

é um aumento da potência dissipada e uma redução no tamanho da caixa delimitadora e no atraso do caminho crítico. O objetivo de economizar energia e a necessidade de não perder muito desempenho nas demais métricas faz com que um valor intermediário seja escolhido. Dessa forma, $k_p = 0.8$ é utilizado nos demais experimentos.

4.4.2 Avaliação do posicionamento

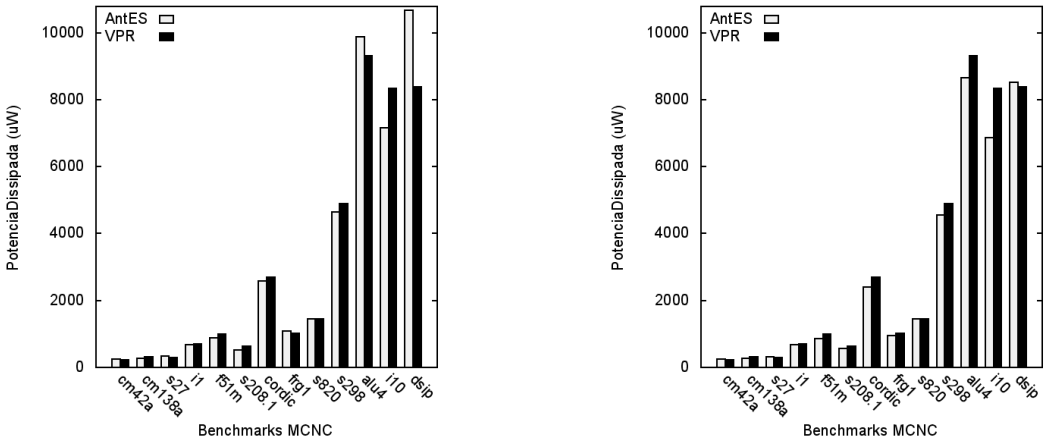
Inicialmente, o posicionamento foi isolado da retemporização para avaliar seu impacto individual no desempenho, em especial no consumo de energia de um FPGA. A eficiência do posicionamento **AntES** depende dos parâmetros k_d, k_p e P_{gen} discutidos na seção anterior e também da função de atratividade, representada como f nas Equações 3.1 e 3.2. Duas funções de atratividade são utilizadas na comparação com o VPR e essas funções têm como componente principal as Equações 4.3 e 4.4.

$$\Delta V = \lambda \,.\, (\Delta T) \;+\; (1 - \lambda) \,.\, (\Delta W) \tag{4.3}$$

$$\Delta V = \lambda \, . \, (\Delta T) \; + \; (1 - \lambda) \, . \, [\, (1 - \sigma) \, . \, (\Delta W) \; + \; \sigma \, . \, (\Delta P) \,] \tag{4.4}$$

O posicionamento **AntES** que visa reduzir o atraso do caminho crítico usa a Equação 4.3 como base para sua função de atratividade. Assim, o mesmo é identificado nas tabelas e gráficos seguintes como "$powerflag = 0$". Por outro lado, o posicionamento voltado a reduzir a potência dissipada, baseia-se na Equação 4.4 e leva em consideração também a variação da potência dissipada ΔP. Por sua vez, esta alternativa é encontrada nas tabelas e gráficos seguintes como "$powerflag = 1$".

A Figura 4.6 ilustra a potência dissipada no FPGA para diferentes circuitos *benchmarks* comparando o posicionamento **AntES** e o VPR. O Gráfico 4.6(a) mostra os resultados quando se utiliza uma função de atratividade orientada ao caminho crítico e o Gráfico 4.6(b) quando se utiliza uma função focando na potência dissipada. Os *benchmarks* estão dispostos em ordem crescente de tamanho, da esquerda para a direita.



(a) $powerflag = 0.0$ (b) $powerflag = 1.0$

Figura 4.6: A potência dissipada no VPR e na **AntES**.

Na abordagem inicial, considerando somente o caminho crítico, o posicionamento **AntES** tem uma potência dissipada em 3% superior ao VPR. Porém, observa-se que a influência é distinta para cada *benchmark* e o desempenho ruim no posicionamento do *benchmark* dsip foi a causa dessa média superior. A inclusão do componente referente ao custo da potência dinâmica, resulta em uma potência dissipada cerca de 8% inferior à obtida pelo VPR, uma melhora de aproximadamente 11% na eficiência em energia entre as duas soluções **AntES** de posicionamento. Esses resultados mostram que a alteraçao na função de atratividade causou uma mudança no foco dos agentes. Mesmo trabalhando de forma distribuída e independente, eles colaboraram entre si para alcançar o objetivo de reduzir a potência dissipada.

A Tabela 4.9 mostra os resultados das duas abordagens de solução **AntES** normalizadas pelo resultado obtido pelo VPR, seguindo o cálculo mostrado na Equação 4.5. Nessa

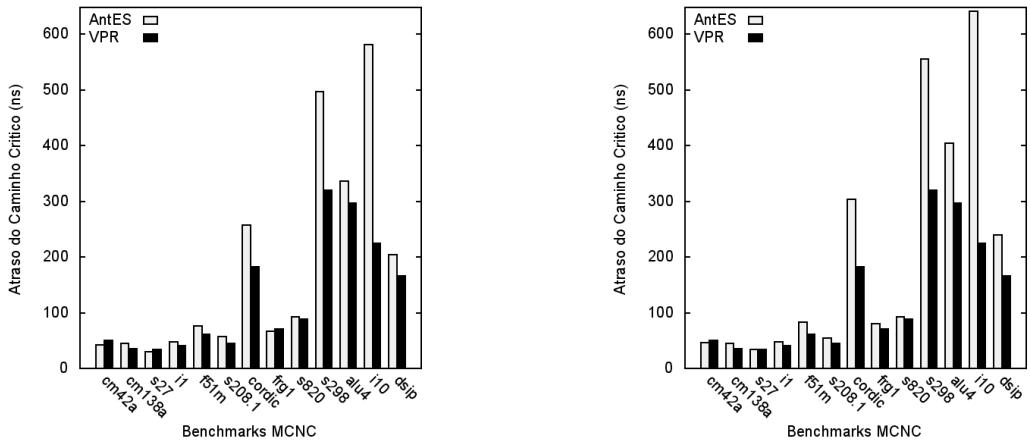
Tabela 4.9: Proporção da potência dissipada para cada categoria.

Categorias	$powerflag = 0$	$powerflag = 1$
pequenos	0.97	0.93
médios	0.96	0.92
grandes	1.07	0.92
geral	1.03	0.92

equação, P_f representa a proporção para a solução $powerflag = f$ onde $f \in 0, 1$. O resultado R obtido pela solução **AntES** f para a métrica x, denotado por $R_{\mathbf{AntES}_f,x}$, é dividido pelo resultado R obtido pelo VPR para a métrica x, denotado por $R_{\mathbf{VPR},x}$. Dessa forma, pode-se observar o desempenho das duas funções de atratividade com relação ao VPR e ao mesmo tempo realizar uma comparação entre elas.

$$P_f = \frac{R_{\mathbf{AntES}_f,x}}{R_{\mathbf{VPR},x}} \tag{4.5}$$

Pela Tabela 4.9 pode-se observar que o posicionamento otimizado para o atraso do caminho crítico dissipa menos potência do que o VPR para os *benchmarks* pequenos e médios. No entanto, o comportamento se altera para os *benchmarks* grandes. Já o posicionamento otimizado para reduzir a potência dissipada, mantém um comportamento regular para as diferentes categorias, reduzindo em até 8% a potência dissipada pelo VPR.



(a) $powerflag = 0.0$ (b) $powerflag = 1.0$

Figura 4.7: O atraso do caminho crítico no VPR e na **AntES**.

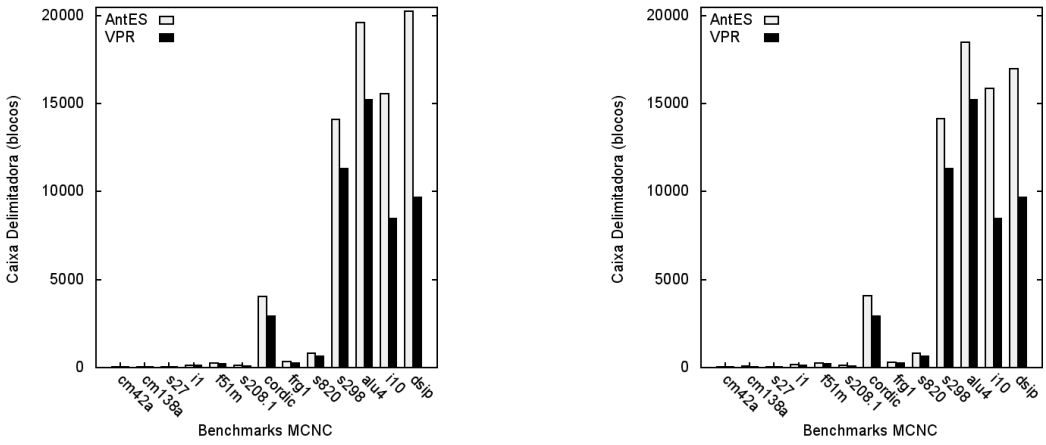
Embora a eficiência em energia tenha aumentado com a técnica de posicionamento proposta, ela tem um impacto negativo no atraso do caminho crítico. A Figura 4.7 mostra a comparação do atraso dos caminhos críticos obtidos pela **AntES** e pelo VPR. A versão da **AntES** que busca otimizar o caminho crítico tem uma média de atraso do caminho crítico 44% superior ao obtido pelo VPR. Na versão otimizada para reduzir a potência dissipada, essa diferença sobe para 62%, pois parte do esforço dedicado a otimizar

o caminho crítico passa a focar na redução da potência dissipada.

Tabela 4.10: Proporção do atraso do caminho crítico para cada categoria.

Categorias	$powerflag = 0$	$powerflag = 1$
pequenos	1.08	1.14
médios	1.37	1.54
grandes	1.63	1.86
geral	1.44	1.62

A Tabela 4.10 mostra uma visão por categorias de *benchmarks* das diferenças de desempenho das soluções **AntES** normalizadas pela solução obtida pelo VPR através da Equação 4.5. A diferença entre os resultados das abordagens de posicionamento **AntES** aumenta com o tamanho dos *benchmarks*. Concentrar parte dos esforços na redução da potência dissipada ($powerflag = 1$), aumenta o atraso do caminho crítico em cerca de 50%.



(a) $powerflag = 0.0$ (b) $powerflag = 1.0$

Figura 4.8: A caixa delimitadora no VPR e na **AntES**.

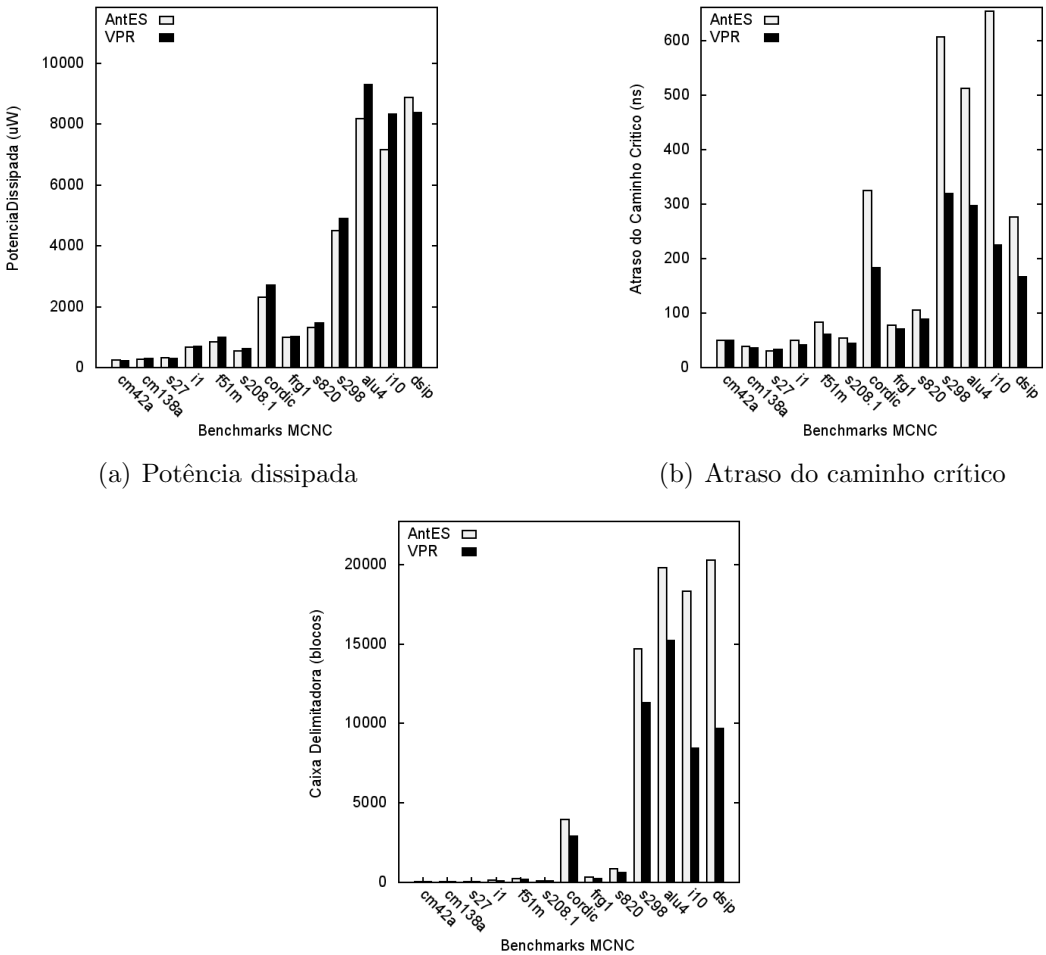
De forma similar, pode-se observar na Figura 4.8 que a caixa delimitadora é maior para a solução **AntES**. No entanto, a Tabela 4.11 mostra que o foco na redução da potência dissipada, com $powerflag = 1$, reduz a caixa delimitadora se comparada à otimizada para o atraso do caminho crítico, $powerflag = 0$. Essa redução ocorre nos *benchmarks* grandes, levando a uma diminuição na caixa delimitadora acumulada para todas as categorias (geral). Isso ocorre porque a potência dissipada, além de depender da taxa de transição, está ligada também ao tamanho da caixa delimitadora de cada interconexão.

A abordagem que inspirou esse trabalho baseia-se no comportamento de formigas realizando o agrupamento de objetos de diferentes naturezas. Nas duas funções de atratividade discutidas anteriormente, cujos resultados são apresentados como $powerflag = 0$

Tabela 4.11: Proporção da caixa delimitadora para cada categoria.

Categorias	$powerflag = 0$	$powerflag = 1$
pequenos	1.19	1.24
médios	1.27	1.28
grandes	1.66	1.54
geral	1.54	1.45

e $powerflag = 1$, as métricas e suas variações durante o algoritmo são usadas na determinação da "atratividade" de cada posição. Uma outra opção para a função de atratividade é a proximidade das interconexões de um determinado bloco. Isso leva os agentes a realizarem o agrupamento de blocos que estão conectados.



(a) Potência dissipada

(b) Atraso do caminho crítico

(c) Caixa delimitadora

Figura 4.9: Posicionamento via agrupamento

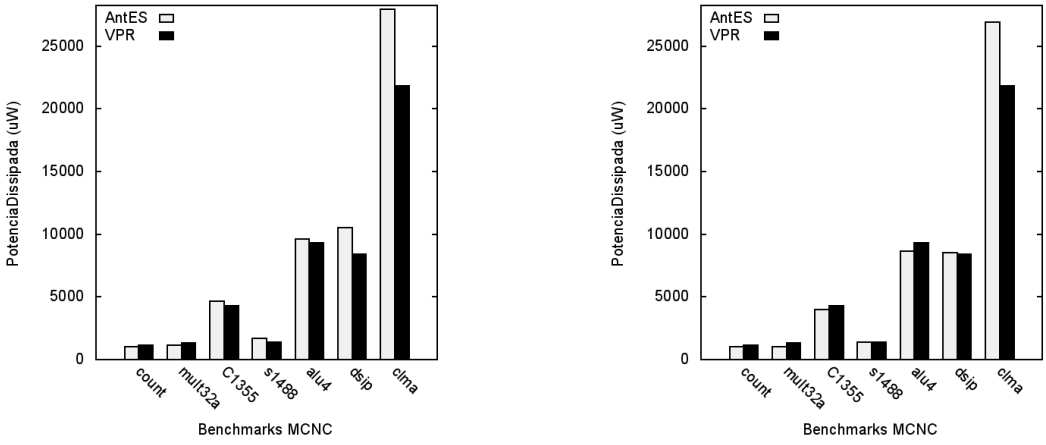
A Figura 4.9 e a Tabela 4.12 mostram o desempenho do posicionamento usando essa função de atratividade. Apesar de ter um ganho de 8% na potência dissipada, apresenta uma perda de 77% no atraso do caminho crítico. Dessa forma, a função de atratividade orientada a potência dissipada ($powerflag = 1$), cujo comportamento é observado na

Figura 4.6, obteve um desempenho superior, tendo ganhos similares em potência dissipada com perdas menores no atraso do caminho crítico e no tamanho da caixa delimitadora.

Tabela 4.12: Proporção dos valores das métricas obtidas utilizando agrupamento normalizadas pelo resultado do VPR.

	$agrupamento/VPR$
Potência dissipada	0.92
Caixa delimitadora	1.61
Atraso do caminho crítico	1.77

Para avaliar o posicionamento da técnica **AntES** no contexto de redes corporais, são usados os *benchmarks* da Tabela 4.4. Desses *benchmarks*, dois são da categoria médio e cinco da categoria grande, dentre eles o maior *benchmark* MCNC, o clma. A Figura 4.10 ilustra a potência dissipada no FPGA para diferentes circuitos *benchmarks* comparando o posicionamento **AntES** e o VPR através de *benchmarks* representativos de redes corporais. O Gráfico 4.11(a) mostra os resultados quando se utiliza uma função de atratividade orientada ao caminho crítico e o Gráfico 4.11(b) quando se utiliza uma função focando na potência dissipada.

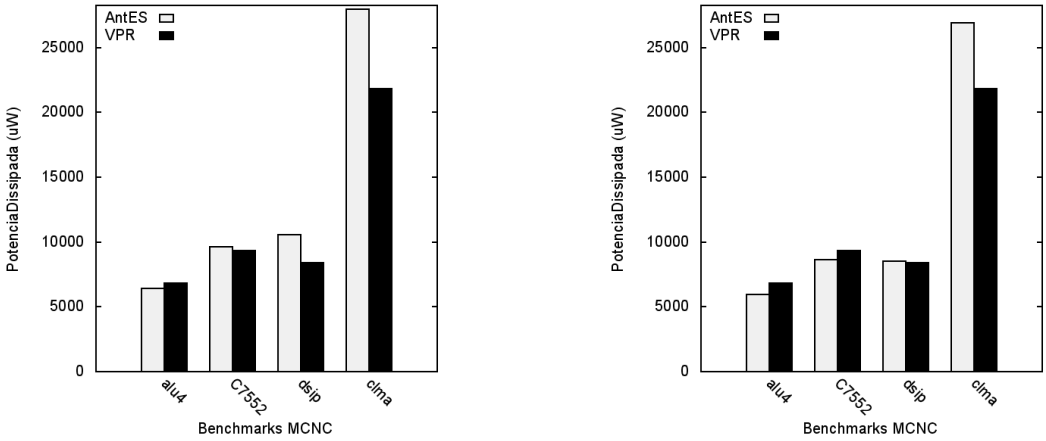


(a) $powerflag = 0.0$ (b) $powerflag = 1.0$

Figura 4.10: A potência dissipada no VPR e na **AntES** no contexto de BANs.

Observa-se que a **AntES** não se comporta como o esperado para os *benchmarks* grandes e sequenciais. Isso acontece porque soluções que otimizam o caminho crítico, como o VPR, deixam os registradores melhor distribuídos, apresentando menos *gliching*. A Figura 4.11 contém um *benchmark* não pertencente ao conjunto inicial, o maior *benchmark* combinacional, C7552. Como ilustra a Figura 4.11, a **AntES** reduz a potência dissipada para os circuitos combinacionais C7552 e alu4, mas tem um efeito limitado nos sequenciais dsip e clma.

A Tabela 4.13 mostra os resultados das duas abordagens de solução **AntES** normalizadas pelo resultado obtido pelo VPR. O posicionamento otimizado para o atraso do



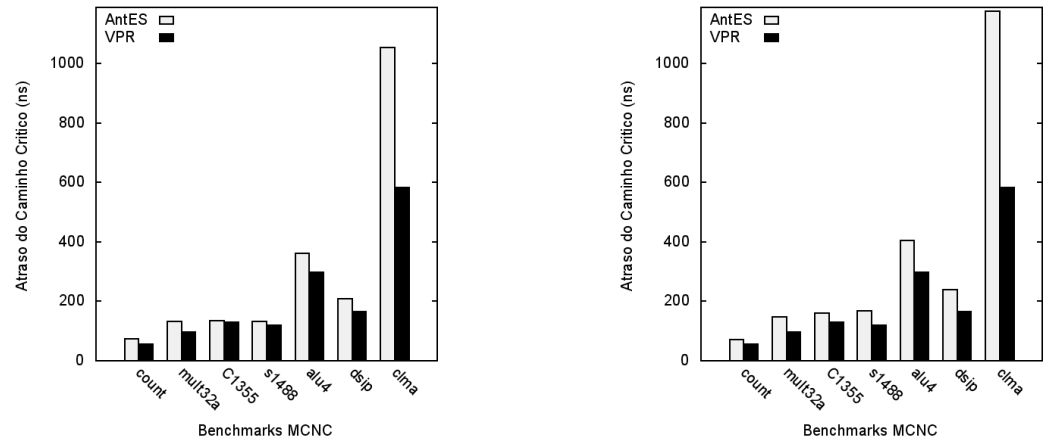
(a) $powerflag = 0.0$ (b) $powerflag = 1.0$

Figura 4.11: A potência dissipada para circuitos combinacionais e sequenciais

caminho crítico apresenta uma potência dissipada igual ou maior do que a VPR. O posicionamento otimizado para reduzir a potência dissipada apresenta um desempenho 11% superior considerando *benchmaks* médios, mas aumenta em 11% a potência dissipada nos *benchmarks* grandes resultando em um aumento global de 8%.

Tabela 4.13: Proporção da média dos valores da potência dissipada para cada categoria em relação ao VPR.

Categorias	$powerflag = 0$	$powerflag = 1$
médios	1	0.89
grandes	1.21	1.11
geral	1.19	1.08



(a) $powerflag = 0.0$ (b) $powerflag = 1.0$

Figura 4.12: O atraso do caminho crítico no VPR e na **AntES**.

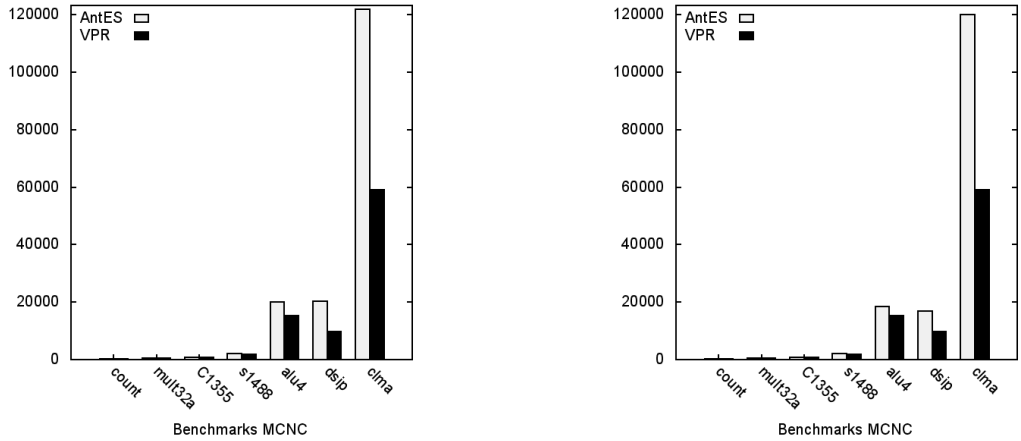
A Figura 4.12 mostra uma comparação do atraso dos caminhos crítico obtido pela **AntES** e pelo VPR. A versão da **AntES** que busca otimizar o caminho crítico tem

uma média de atraso do caminho crítico 44% superior ao obtido pelo VPR. Na versão otimizada para reduzir a potência dissipada, essa diferença sobe para 63%, pois parte do esforço antes dedicado a otimizar o caminho crítico passa a focar na redução da potência dissipada. Esse aumento no atraso do caminho crítico, no entanto, não compromete seu desempenho em aplicações de tempo real, pois possui uma velocidade de processamento ainda superior a implementações em *software*.

Tabela 4.14: Proporção da média dos valores de atraso do caminho crítico para cada categoria em relação ao VPR.

Categorias	$powerflag = 0$	$powerflag = 1$
médios	1.20	1.34
grandes	1.50	1.70
geral	1.44	1.63

A Tabela 4.14 mostra uma visão por categorias de *benchmarks* das diferenças de desempenho das soluções **AntES** normalizadas pela solução obtida pelo VPR. A diferença entre os resultados das abordagens de posicionamento **AntES** aumenta com o tamanho dos *benchmarks*. Concentrar parte dos esforços na redução da potência dissipada ($powerflag = 1$), aumenta o atraso do caminho crítico em cerca de 50% para os *benchmarks* grandes.



(a) $powerflag = 0.0$ (b) $powerflag = 1.0$

Figura 4.13: A caixa delimitadora no VPR e na **AntES**.

A caixa delimitadora é maior quando se a solução **AntES**, da mesma forma que o atraso do caminho crítico, como ilustra a 4.13 O tamanho da caixa delimitadora chega a ser 83% maior que a obtida pelo VPR, como descreve a Tabela 4.15. Esse resultado mostra que alguns blocos são posicionados distantes dos seus vizinhos, pois são posicionados próximos somente os blocos cujas estimativas de atividade das interconexões são grandes. No entanto, considerando as duas funções de atratividade **Antes**, observa-se que o foco

na redução da potência dissipada reduz a caixa delimitadora nos *benchmarks* grandes, o que leva a uma redução de 7% entre as duas alternativas.

Tabela 4.15: Proporção da média dos valores da caixa delimitadora para cada categoria em relação ao VPR.

Categorias	$power flag = 0$	$power flag = 1$
médios	1.27	1.28
grandes	1.91	1.83
geral	1.90	1.83

Os resultados mostram que a *AntES* reduz a potência dissipada para a maioria dos *benchmarks* utilizados, salvo para alguns *benchmarks* grandes e sequenciais. No entanto, ela causa um aumento no atraso do caminho crítico e no tamanho da caixa delimitadora. A potência dissipada pode ser reduzida para os *benchmarks* sequenciais através da redução no *glitching*. Dessa forma, também neste contexto o posicionamento intercalado à retemporização se mostra interessante.

4.5 Conclusão

Este capítulo apresentou os resultados obtidos pelo algoritmo de posicionamento da **AntES**. As estimativas foram obtidas através do modelo de energia do VPR em um cenário genérico e em um cenário fictício de redes corporais, diferenciados pelo tipo e tamanho dos circuitos *benchmarks* utilizados. Os parâmetros arquiteturais dos FPGAs foram aqueles considerados mais adequados segundo a literatura. Os cenários foram criados a partir de *benchmarks* MCNC, tradicionalmente utilizados na avaliação de algoritmos implementados no fluxo CAD de FPGAs.

Nas avaliações, os resultados do posicionamento da técnica **AntES** foi comparado aos obtidos pelo VPR. Essa avaliação mostrou uma redução na potência dissipada para a maioria dos *benchmarks* utilizados, salvo para alguns *benchmarks* grandes e sequenciais. Porém, essa redução foi acompanhada por um aumento no atraso do caminho crítico e no tamanho da caixa delimitadora.

CAPÍTULO 5

CONCLUSÕES

Os FPGAs consomem mais energia que os microcontroladores e os circuitos integrados de aplicação específica (*Application-Specific Integrated Circuits - ASICs*) devido aos seus componentes reconfiguráveis. Isso dificulta sua adoção na implementação de sistemas em dispositivos com restrições de recursos, como nós sensores. Para que sua utilização nesses ambientes seja possível, são necessárias técnicas para reduzir o seu consumo de energia.

O mapeamento de circuitos desenvolvidos em linguagens de descrição de *hardware* para um FPGA é alcançado através de ferramentas CAD. Essas ferramentas são responsáveis pelas diversas etapas pelas quais essa descrição de alto nível passa antes de ser de fato mapeada em um circuito físico. Todas as etapas do fluxo CAD oferecem oportunidades para processos automatizados de otimização no consumo de energia.

Uma das técnicas implementadas em diferentes níveis do fluxo CAD é a segmentação. Sabe-se que a inserção de registradores no circuito reduz o *glitching*, economizando energia. A segmentação e retemporização, juntas, segmentam o circuito de forma automática. Ao contrário do acontece com ASICs, a retemporização em FPGAs só é eficiente quando implementada na fase de posicionamento, de forma simultânea a este. Técnicas de posicionamento e retemporização simultâneos foram propostas, com foco na velocidade de processamento, mas os movimentos de retemporização não são de fato intercalados em pequena granularidade aos de posicionamento.

Este trabalho apresentou a técnica de posicionamento, segmentação e retemporização **AntES** para economizar energia em FPGAs. Nesta técnica, após uma geração inicial de registradores, agentes de posicionamento e retemporização trabalham de forma simultânea para posicionar e segmentar o circuito de forma eficiente. Os agentes trabalham de forma distribuída e independente. Da mesma forma que as formigas nos quais são inspirados, eles interagem indiretamente entre si e suas interações possibilitam que o posicionamento e a retemporização sejam realizados.

Um circuito bem posicionado representa para os agentes uma grande quantidade de ferormônios. As operações de pegar e deixar um componente de circuito (bloco lógico ou registrador), são tomadas baseadas nessa variação de ferormônios, cuja intensidade determina a atratividade de uma posição. O posicionamento é voltado reduzir a potência dissipada através da identificação de interconexões cujas estimativas de atividade são mais altas. Já a retemporização busca o período de *clock* mínimo para um circuito já segmentado. Essa segmentação, além de aumentar a velocidade do circuito, reduz o *glitching*. Dessa forma, a segmentação reduz a potência dinâmica dissipada.

Os resultados da nova solução foram comparados aos obtidos pela versão normal do VPR utilizando um subconjunto dos *benchmarks* MCNC e cenários de avaliação que consideram as características das redes corporais. A avaliação dos resultados mostram que o posicionamento da técnica **AntES**, quando otimizado para economizar energia, reduz a potência dissipada pela maioria dos *benchmarks*, com notável exceção de alguns *benchmarks* grandes e sequenciais como dsip e clma. No entanto, o atraso do caminho crítico é prejudicado por essa solução, pois parte do esforço de otimização deixa de lado o atraso do caminho crítico.

Observa-se oportunidades de melhora no desempenho do posicionamento usando essa técnica, pois sua característica distribuída herdadas da natureza, oferecem flexibilidade permitindo o teste de diferentes abordagens. Além de funções de atratividade distintas, a solução pode ser combinada com algoritmos evolutivos ou com o próprio *Simulated Annealing* já empregado em algoritmos existentes.

5.1 Trabalhos Futuros

Esta seção apresenta os trabalhos futuros, que estendem ou derivam desta pesquisa. O primeiro deles complementa a avaliação do desempenho da retemporização, destacando as dificuldades encontradas na sua implementação. A técnica apresentada pode ser incluída como parte de uma ferramenta CAD largamente usada no mercado de FPGAs, como o Quartus II da Altera. Assim, pode-se avaliar a eficácia da solução apresentada através de uma experimentação em um FPGA. Também são apresentadas as experimentações que podem ser realizadas.

Retemporização

Durante a implementação da técnica **AntES** no VPR foram encontradas algumas dificuldades que impossibilitaram avaliação da retemporização. O VPR é implementado em linguagem C e suas estruturas de dados restringem o modo como novas soluções podem ser adicionadas. Ele possui estruturas que representam a matriz de blocos do FPGA, o grafo de interconexões cujos nós são elementos físicos como pinos de entrada e saída que podem ser conectados entre si, dentre outras. A técnica de posicionamento apresenta uma implementação imediata neste contexto, pois não sofre imposições das estruturas. Neste caso, mudam as regras de como os blocos são posicionados. Já a retemporização, implementada em nível de posicionamento, exige meios de criar e remover registradores para que as informações de tempo possam ser atualizadas e mais precisas. No entanto, a movimentação dos agentes que resulta na criação e remoção de registradores, necessita de estruturas mais flexíveis que possam ser alteradas com mais facilidade e menos efeitos colaterais. O grafo de interconexões, por exemplo, permite apenas a inspeção de seus nós

caminhando em direção as saídas, criando a necessidade de outras estruturas de suporte para prover informações originalmente não disponíveis. Alterações nesse grafo, por exemplo um movimento de retemporização para frente, devem ser propagadas até as saídas e ainda atualizadas na matriz que representa o FPGA.

Apesar dessas dificuldades, a implementação da retemporização em nível de posicionamento é possível e um importante trabalho futuro a ser implementado no fluxo CAD de FPGAs. A especificação desenvolvida neste trabalho, apresenta uma solução distribuída onde a independência entre os agentes permite que os movimentos possam ser intercalados aos de posicionamento. Isso possibilita uma retemporização mais precisa sem o comprometimento do posicionamento.

Experimentação em um FPGA

Os resultados mostrados no capítulo de avaliação foram obtidos através de estimativas do modelo de energia implementado pelo VPR. Este modelo é bastante preciso para um modelo analítico, por isso é utilizado em muitos trabalhos com o fluxo CAD de FPGAs. Porém, a validação dos resultados em uma experimentação real, aproxima o cenário simplificado utilizado de um cenário mais complexo onde circuitos funcionais e FPGAs mais modernos são utilizados. Nos FPGAs atuais, muitas operações frequentes, como multiplicação e adição, são efetuadas através de circuitos específicos embarcados nos FPGAs. Esses detalhes podem resultar em diferenças de desempenho cujo impacto na solução deve ser avaliado.

Avaliação do FPGA em redes corporais

A implementação de FPGAs em redes corporais seria melhor destacada em uma experimentação na qual um FPGA controla um ou mais sensores espalhados no corpo de um paciente. Um trabalho futuro importante é a implementação de um FPGA como um agregador (*clusterhead*) numa rede corporal. Esse nó, além das funções de sensoriamento, seria responsável também por agregar dados vindos dos demais nós sensores. Para isso, circuitos que implementam as funções do *clusterhead* são codificados em uma linguagem de descrição de *hardware*. Estes circuitos são mapeados no FPGA passando pelas etapas de posicionamento e retemporização descritos neste trabalho. Dessa forma, seria possível avaliar o consumo de energia do FPGA durante o monitoramento do paciente. Neste contexto, podem ser realizadas uma avaliação local, através das métricas já apresentadas, e uma avaliação global considerando métricas da rede como um todo.

BIBLIOGRAFIA

[1] K. Lorincz, D. J. Malan, T. R. F. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, M. Welsh, and S. Moulton. "Sensor networks for emergency response: challenges and opportunities". *Pervasive Computing, IEEE*, volume 3(número 4):páginas 16–23, outubro 2004.

[2] M. A. Hanson, H. C. Powell, A. T. Barth, K. Ringgenberg, B. H. Calhoun, J. H. Aylor, and J. Lach. "Body Area Sensor Networks: Challenges and Opportunities". *Computer*, volume 42(número 1):páginas 58–65, janeiro 2009.

[3] I. Kuon and J. Rose. "Measuring the gap between FPGAs and ASICs". In *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, pages 21–30, New York, NY, USA, 2006. ACM.

[4] B. Scheuermann. *"Ant Colony Optimization on Runtime Reconfigurable Architectures"*. PhD thesis, Karlsruhe Institute of Technology, 2005.

[5] S. Commuri, V. Tadigotla, and M. Atiquzzaman. "Reconfigurable Hardware Based Dynamic Data Aggregation in Wireless Sensor Networks". *International Journal of Distributed Sensor Networks*, volume 4(número 2):páginas 194–212, 2008.

[6] B.A. Draper, J.R. Beveridge, A.P.W. Bohm, C. Ross, and M. Chawathe. "Accelerated image processing on FPGAs". *IEEE Transactions on Image Processing*, volume 12(número 12):páginas 1543–1551, dezembro 2003.

[7] Fan Y. and M. Paindavoine. "Implementation of an RBF neural network on embedded systems: real-time face tracking and identity verification". *IEEE Transactions on Neural Networks*, volume 14(número 5):páginas 1162–1175, setembro 2003.

[8] T. Ahola, P. Korpinen, J. Rakkola, T. Ramo, J. Salminen, and J. Savolainen. "Wearable FPGA Based Wireless Sensor Platform". pages 2288–2291, agosto 2007.

[9] J. Lamoureux and S. J. E. Wilton. "On the Interaction Between Power-aware FPGA CAD Algorithms". In *International Conference on Computer Aided Design*, pages 701–708, 2003.

[10] Q. Wang, S. Gupta, and J. H. Anderson. "Clock Power Reduction for Virtex-5 FPGAs". In *Proceeding of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 13–22, New York, NY, USA, 2009. ACM.

[11] J. Lamoureux and W. Luk. "An Overview of Low-Power Techniques for Field-Programmable Gate Arrays". In *NASA/ESA Conference on Adaptive Hardware and Systems*, pages 338–345, junho 2008.

[12] K. K. W. Poon, S. J. E. Wilton, and A. Yan. "A Detailed Power Model for Field-Programmable Gate Arrays". *ACM Transactions on Design Automation of Electronic Systems*, volume 10(número 2):páginas 279–302, 2005.

[13] J. Lamoureux, G. Lemieux, and S. Wilton. "GlitchLess: Dynamic Power Minimization in FPGAs Through Edge Alignment and Glitch Filtering". *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, volume 16(número 11):páginas 1521–1534, novembro 2008.

[14] M. Dorigo, V. Maniezzo, and A. Colorni. "Ant System: Optimization by a Colony of Cooperating Agents". *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, volume 26(número 1):páginas 29–41, fevereiro 1996.

[15] I. Kuon, R. Tessier, and J. Rose. "FPGA Architecture: Survey and Challenges". *Foundations and Trends in Electronic Design Automation*, volume 2(número 2):páginas 135–253, 2008.

[16] E. Ahmed. "The Effect of Logic Block Granularity on Deep-Submicron FPGA Performance and Density". Master's thesis, University of Toronto, Department of Electrical and Computer Engineering, 2001.

[17] V. Betz and J. Rose. "How Much Logic Should Go in an FPGA Logic Block?". *IEEE Design and Test of Computers*, volume 15(número 1):páginas 10–15, 1998.

[18] A. Roopchansingh and J. Rose. "Nearest Neighbour Interconnect Architecture in Deep Submicron FPGAs". In *In IEEE Custom Integrated Circuits Conference*, pages 59–62, 2002.

[19] S. Sivaswamy, G. Wang, C. Ababei, K. Bazargan, R. Kastner, and E. Bozorgzadeh. "HARP: Hard-Wired Routing Pattern FPGAs". In *Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays*, pages 21–29, New York, NY, USA, 2005. ACM.

[20] D. P. Singh and S. D. Brown. "The Case for Registered Routing Switches in Field-Programmable Gate Arrays". In *Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays*, pages 161–169, New York, NY, USA, 2001. ACM.

[21] Xilinx. "Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet". `http://www.xilinx.com/support/documentation/data_sheets/ds083.pdf`, abril 2010.

[22] D. Chen, J. Cong, and P. Pan. "FPGA Design Automation: A Survey". *Foundations and Trends in Electronic Design Automation*, volume 1(número 3):páginas 139–169, 2006.

[23] J. I. Hillawi and K. R. Bennett. "EDIF - An Overview". *Computer-Aided Engineering Journal*, volume 3(número 3):páginas 102–107, junho 1986.

[24] V. Betz. *"Placement for General Purpose FPGAs"*, chapter 14. Reconfigurable Computing. Morgan Kauffman, 2007.

[25] M. El-Abd, H. Hassan, M. Anis, M. S. Kamel, and M. Elmasry. "Discrete Cooperative Particle Swarm Optimization for FPGA Placement". *Applied Soft Computing*, volume 10(número 1):páginas 284–295, 2010.

[26] S. Kirkpatrick, Jr. Gelatt, C. D., and M. P. Vecchi. "Optimization by Simulated Annealing". *Science*, volume 220(número 4598):páginas 671–680, 1983.

[27] A. Marquardt, V. Betz, and J. Rose. "Timing-Driven Placement for FPGAs". In *Proceedings of the 2000 ACM/SIGDA eighth international symposium on Field programmable gate arrays*, pages 203–213, New York, NY, USA, 2000. ACM.

[28] David A. Patterson and John L. Hennessy. *"Computer Organization and Design, Fourth Edition, Fourth Edition: The Hardware/Software Interface (The Morgan Kaufmann Series in Computer Architecture and Design)"*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4th edition, 2008.

[29] C. E. Leiserson and J. B. Saxe. "Retiming Synchronous Circuitry". *Algorithmica*, volume 6(1-6):páginas 5–35, Junho 1991.

[30] C. E. Leiserson and J. B. Saxe. "Optimizing Synchronous Systems". In *22nd Annual Symposium on Foundations of Computer Science*, pages 23–36, 28-30 1981.

[31] N. Weaver, Y. Markovskiy, Y. Patel, and J. Wawrzynek. "Post-placement C-slow Retiming for the Xilinx Virtex FPGA". In *Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on Field-Programmable Gate Arrays*, pages 185–194, New York, NY, USA, 2003. ACM.

[32] D. P. Singh and S. D. Brown. "Integrated Retiming and Placement for Field Programmable Gate Arrays". In *Proceedings of the 2002 ACM/SIGDA Tenth International*

Symposium on Field-Programmable Gate Arrays, pages 67–76, New York, NY, USA, 2002. ACM.

[33] K. Eguro and S. Hauck. "Simultaneous Retiming and Placement for Pipelined Netlists". In *16th International Symposium on Field-Programmable Custom Computing Machines.*, pages 139–148, abril 2008.

[34] J. Lamoureux and S.J.E. Wilton. "Activity Estimation for Field-Programmable Gate Arrays". In *International Conference on Field Programmable Logic and Applications*, pages 1–8, 2006.

[35] S.-H. Chow, Y.-C. Ho, T. Hwang, and C. L. Liu. "Low Power Realization of Finite State Machines—a Decomposition Approach". *ACM Transactions on Design Automation of Electronic Systems*, volume 1(número 3):páginas 315–340, 1996.

[36] Y. Zhang, J. Roivainen, and A. Mammela. "Clock-Gating in FPGAs: A Novel and Comparative Evaluation". In *Proceedings of the 9th EUROMICRO Conference on Digital System Design*, pages 584–590, 2006.

[37] J. Lamoureux and S. J. E. Wilton. "FPGA Clock Network Architecture: Flexibility vs. Area and Power". In *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, pages 101–108, New York, NY, USA, 2006. ACM.

[38] J. Lamoureux and S.J.E. Wilton. "Clock-Aware Placement for FPGAs". In *International Conference on Field Programmable Logic and Applications*, pages 124–131, 2007.

[39] P. Jamieson, W. Luk, S.J.E. Wilton, and G.A. Constantinides. "An energy and power consumption analysis of FPGA routing architectures". In *International Conference on Field-Programmable Technology, 2009. FPT 2009.*, pages 324–327, 2009.

[40] Latanya Sweeney. "That's AI? A History and Critique of the Field". *Mellon University, School of Computer Science, Pittsburgh*, pages 03–106, 2003.

[41] A. J. Greensted and A. M. Tyrrell. "An Endocrinologic-Inspired Hardware Implementation of a Multicellular System". *Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware*, page página 245, 2004.

[42] G. Tempesti, D. Roggen, E. Sanchez, Y. Thoma, R. Canham, and A. M. Tyrrell. "Ontogenetic Development and Fault Tolerance in the POEtic Tissue". In *5th International Conference on Evolvable Systems (ICES'2003)*, pages 141–152, 2003. Tyrrell, A. M. and Haddow, P. C. and Torresen, J. (eds.).

[43] J. Eriksson, O. Torres, A. Mitchell, G. Tucker, K. Lindsay, D. M. Halliday, J. Rosenberg, J. M. Moreno, and A. E. P. Villa. "Spiking Neural Networks for Reconfigurable POEtic Tissue". In *Evolvable Systems: From Biology to Hardware*, volume volume 2606 of *Lecture Notes in Computer Science*, pages páginas 165–173. Springer Berlin / Heidelberg, 2003.

[44] G. Tempesti, D. Mange, E. Petraglio, A. Stauffer, and Y. Thoma. "Developmental Processes in Silicon: An Engineering Perspective". In *Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware*, page 265, Washington, DC, USA, 2003. IEEE Computer Society.

[45] J. M. Moreno, Y. Thoma, and E. Sanchez. "POEtic: A Prototyping Platform for Bio-inspired Hardware". In J.M. Moreno, J. Madrenas, and J. Cosp, editors, *Evolvable Systems: From Biology to Hardware (ICES 2005)*, volume 3637 of *LNCS*, pages 177–187, Berlin Heidelberg, 2005. Springer-Verlag.

[46] E. Sanchez, D. Mange, M. Sipper, M. Tomassini, A. Perez-Uribe, and A. Stauffer. "Phylogeny, Ontogeny, and Epigenesis: Three Sources of Biological Inspiration for Softening Hardware". In *Evolvable Systems: From Biology to Hardware*, volume volume 1259 of *Lecture Notes in Computer Science*, pages páginas 33–54. Springer Berlin / Heidelberg, 1997.

[47] G. Tempesti, D. Roggen, E. Sanchez, Y. Thoma, R. Canham, A. Tyrrell, and J. Moreno. "A POEtic architecture for bio-inspired hardware". In *ICAL 2003: Proceedings of the eighth international conference on Artificial life*, pages páginas 111–115, Cambridge, MA, USA, 2003. MIT Press.

[48] A. Upegui, Y. Thoma, E. Sanchez, A. Perez-Uribe, J. M. Moreno, J. Madrenas, and G. Sassatelli. "The PERPLEXUS Bio-inspired Hardware Platform: A Flexible and Modular Approach". *International Journal of Knowledge-based and Intelligent Engineering Systems*, volume 12:páginas 201–212, Agosto 2008.

[49] E. Bonabeau, M. Dorigo, and G. Theraulaz. *"Swarm Intelligence: From Natural to Artificial Systems"*. Oxford University Press, USA, 1 edition, Setembro 1999.

[50] J. L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chrétien. "The Dynamics of Collective Sorting Robot-Like Ants and Ant-Like Robots". In *Proceedings of the First International Conference on Simulation of Adaptive Behavior on From Animals to Animats*, pages 356–363, Cambridge, MA, USA, 1990. MIT Press.

[51] A. Forestiero, C. Mastroianni, and M. Meo. "Self-Chord: A Bio-inspired Algorithm for Structured P2P Systems". In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 44–51, Washington, DC, USA, 2009. IEEE Computer Society.

[52] Altera. "Quartus II". `http://www.altera.com/products/software/quartus-ii/subscription-edition/qts-se-index.html`, abril 2010.

[53] Xilinx. "ISE". `http://www.xilinx.com/support/download/index.htm`, abril 2010.

[54] Actel. "Libero". `http://www.actel.com/products/software/libero/`, abril 2010.

[55] Mentor Graphics. "ModelSim". `http://model.com/`, abril 2010.

[56] J. Luu, I. Kuon, P. Jamieson, T. Campbell, A. Ye, W. M. Fang, and J. Rose. "VPR 5.0: FPGA cad and architecture exploration tools with single-driver routing, heterogeneity and process scaling". In *Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays*, pages 133–142, New York, NY, USA, 2009. ACM.

[57] S. Yang. "Logic Synthesis and Optimization Benchmarks User Guide – Version 3.0", 1991.