Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Pós-Graduação em Ciência da Computação

RSCM
Controlador de Configurações para Sistemas de Hardware Reconfigurável

por
Ewerson Luiz de Souza Carvalho

Dissertação apresentada como requisito parcial à obtenção do grau de mestre em Ciência da Computação

Orientador: Prof. Dr. Ney Laert Vilar Calazans

Porto Alegre, março de 2004

Av. Ipiranga, 6681 - Prédio 16
CEP 90619-900 - Porto Alegre - RS - Brasil
http://www.pucrs.br/biblioteca

Fone: 0 (xx)51 3320-3544
Fax: 0 (xx)51 3320-3548
e-mail: bceadm@pucrs.br

Para João Marcelo Vaz Carvalho.

"Until the philosophy which holds one race
Superior and another inferior
Is finally and permanently discredited and abandoned
Everywhere is war, me say war"

Robert Nesta Marley (1945-1981)

Agradecimentos

Agradeço ao CNPq por financiar este trabalho e o PPGCC por fornecer todo suporte necessário. Só uma vez faltaram alguns fios, mas tudo bem. Agradeço todos funcionários da secretaria do PPGCC, principalmente ao Leandro que sempre ficava alguns minutinhos a mais trabalhando, esperando que eu acabasse meus trabalhos, quando eram já onze horas da noite.

Agradeço meus eternos amigos: Anderson, Vagner e Richard. Nos distanciamos porque cada um teve de seguir seu rumo, mas as coisas que aprendemos juntos e ensinamos uns aos outros me ajudaram a ultrapassar diversos problemas em minha vida de forma que estaremos sempre juntos.

Aos meus amigos do mestrado e membros do GAPH agradeço pela excelente convivência que me proporcionaram. Nem quero imaginar como seria a rotina diária se vocês não fossem pessoas tão agradáveis. As rodas de chimarão, as cervejadas, as partidas de futebol, os "cafezinhos" do Noriaki, as piadas sem graça o Ost etc.

Agradeço Leonardo Castanheira, amigo desde os tempos de graduação, que me ajudou a entrar no mestrado e que está me ajudando a conseguir um emprego. Pô, o cara é um pai pra mim, e eu não paguei nada por essa dedicação, é coisa de amigo mesmo.

Um agradecimento especial ao meu amigo Edson. Sem você teria sido impossível enfrentar os desafios dessa fase da minha vida que passou. A frase que tu me disse logo nos primeiros dias do mestrado se concretizou. Eu lembro: "as amizades que se iniciarem durante o mestrado serão mais intensas que as anteriores em virtude das dificuldades ultrapassadas". Eu sempre vou contar para todo mundo que foi a Sílvia, tua patroa, que me emprestou dinheiro no dia 26 de Abril de 2003 para eu viajar e ver meu filho recém nascido. Muitas felicidades para vocês dois.

Valeu Fred! Obrigado por ajudar no nosso trabalho. Eu quero que tu saibas que esse trabalho também é teu. Claro que tens que me pagar uma certa quantia por isso.

Obrigado Moraes por não gostar de mim. Eu te admiro muito. Tu és um exemplo a ser seguido. Nem vou te desejar sucesso, porque tu já tens, deixa um pouco para os outros.

Ney, me faltam palavras de agradecimento. Obrigado por acreditar em mim e por me proporcionar a oportunidade conviver com pessoas tão brilhantes. É uma satisfação ter compartilhado esse período contigo, no qual pude observá-lo e aprender muito mais que conceitos relacionados a sistemas dinâmica e parcialmente reconfiguráveis.

Pai! Mãe! Vocês são tão responsáveis por tudo que realizei e realizarei na minha vida. Será que eu estaria aqui ainda se vocês não estivessem comigo todas as vezes que eu

estava doente, me cuidando, me amando. Vocês estão intrinsecamente inclusos nas minhas derrotas e vitórias, que me faltam palavras de agradecimento. A única retribuição que eu posso dar é AMAR muito vocês.

Wagner! Obrigado por fazer parte da minha vida. Sem você a minha vida seria muito solitária. Eu só não gosto de ti porque tu nunca usaste Kichute, aquele tênis horrível que eu usei muito. É bom ter um irmão menor para sacanear. Esquece de tentar me ganhar na força tchê! Vai estudar tchê! Parabéns pela aprovação no vestibular! Na engenharia tu vais ver o que é bom.

Obrigado Lílian, por estar sempre comigo em sentimentos mesmo com a distância física dos últimos anos mas, principalmente por me dar o maior presente que eu já ganhei: o João Marcelo, um guri lindo e saudável. Agradeço aos teus pais, pelo suporte incondicional que fornecem ao nosso filhinho.

João Marcelo, a partir do teu nascimento, a minha vida tomou um rumo diferente. Quando tu estavas na barriga da tua mãe eu não tinha percebido a importância do acontecimento. Mas quando eu entrei naquele quarto e te vi, bem pequenininho, frágil e vulnerável eu percebi da responsabilidade que implicava ser pai. Eu vou te ensinar tudo que eu puder, aprender contigo tudo que eu puder, te cuidar, te AMAR. Sei que nada é perfeito, mas vou ser o melhor que eu posso para que tu possas ser o melhor. Obrigado por mudar o rumo da minha vida. O pai te AMA demais!

Resumo

Sistemas reconfiguráveis onde o hardware pode ser alterado em tempo de execução possuem o potencial para flexibilizar hardware de forma similar à flexibilidade provida pelo uso de software. Eles podem apresentar a vantagem adicional de poderem simultaneamente alcançar melhor desempenho e menor tamanho. Contudo, existem carências em dispositivos de suporte, ferramentas e fluxos de projeto para tais sistemas. Uma das principais carências são métodos eficientes de controle do processo de reconfiguração do hardware. A principal contribuição deste trabalho é a proposta e construção de um controlador de configurações de hardware implementado totalmente em hardware, em contraposição a propostas da literatura, realizadas predominantemente em software. Uma característica importante do controlador implementado é que este é parte do hardware do sistema, tornando o mesmo capaz de se autoreconfigurar, sem recurso a dispositivos de controle externos. O modelo subjacente proposto, denominado RSCM, é genérico para uma certa classe de aplicações e dispositivos, podendo ser implementado em hardware, software ou com um misto de ambos. Ainda no presente trabalho, apresenta-se um resumo do estado da arte em sistemas reconfiguráveis, com ênfase em sistemas dinâmica e parcialmente reconfiguráveis. Propõe-se o arcabouço PaDReH para projeto e gerenciamento destes sistemas. Além disto, alguns critérios de classificação do processo de reconfiguração de sistemas são propostos para auxiliar a compreensão do mesmo.

Abstract

Reconfigurable systems where the hardware can be changed during execution time have the potential to provide hardware with the same flexibility as software. These may, at the same time, achieve better performance and smaller size. However, there is a clear lack of support devices, tools and design flows adequate for such systems. One of the main problems is the unavailability of efficient methods to control the hardware reconfiguration process. The main contribution of this work is the proposal and construction of a hardware reconfiguration controller totally built in hardware. This is different from previous approaches, where software implementations dominate. An important characteristic of the implemented controller is that it is part of the system hardware, making the latter capable of self reconfiguration without resource to external controlling devices. The underlying configuration controller model, named RSCM, is generic within a certain class of applications and devices, and can be implemented in hardware, software or as a mix of both. Also included is a brief review of the state of the art in hardware reconfigurability, putting emphasis in dynamically and partially reconfigurable systems and devices. The PaDReH framework for the design and management of reconfigurable systems is proposed. Besides, some classification criteria for the system reconfiguration process are advanced to help in the understanding of the overall problem.

Sumário

RESUMO **i**

ABSTRACT **iii**

LISTA DE TABELAS **ix**

LISTA DE FIGURAS **xi**

LISTA DE SÍMBOLOS E ABREVIATURAS **xiii**

Capítulo 1: Introdução **1**

1.1 SoCs e Núcleos de propriedade intelectual 1
1.2 Sistemas digitais reconfiguráveis 3
1.2.1 Modelo genérico para sistemas reconfiguráveis 5
Um computador como um sistema reconfigurável 8
Um DISC como um sistema reconfigurável 9
Um contra-exemplo 10
1.3 Motivações 11
1.4 PaDReH - arcabouço para SDRs 14
1.5 Objetivos 15
1.6 Organização do volume 17

Capítulo 2: Reconfiguração dinâmica e parcial **19**

2.1 Classificação do processo de configuração 20
2.1.1 Largura do caminho de dados de configuração 20
2.1.2 Comando do processo de configuração 20
2.1.3 Localização da memória de configuração 20
2.1.4 Tamanho do arquivo de configuração 21
2.1.5 Dimensão dos módulos manipulados na configuração 21
2.1.6 Formato do arquivo de configuração 21
2.2 Suporte para reconfiguração dinâmica e parcial 22
2.2.1 Tecnologias habilitadoras 23

A plataforma V2MB1000 da Memec-Insight . . . 25
2.2.2 Manipulação de arquivos de configuração . . . 28
PARBIT . . . 29
JBits . . . 29
JPG . . . 29
JRTR . . . 30
Ferramentas desenvolvidas no GAPH . . . 30
Alternativas para geração de arquivos de configuração parciais . . . 30
2.2.3 Métodos de RTR dirigidos a núcleos IP . . . 31
Barramento de Palma . . . 31
Fluxo de projeto modular de Lim . . . 32

Capítulo 3: Controladores de configurações 35
3.1 Modelos de controladores de configurações . . . 36
3.1.1 Modelo de Shirazi et al. . . . 36
3.1.2 Modelo de Burns et al. . . . 37
3.1.3 Modelo de Lysaght et al. . . . 39
3.2 Implementações de controladores de configurações . . . 41
3.2.1 Implementação de Curd . . . 42
3.2.2 Implementação de Blodget et al. . . . 43
3.3 Considerações em controladores de configurações . . . 44
3.4 Gerenciamento de recursos em SDRs . . . 47
3.4.1 Estratégias de alocação em dispositivos reconfiguráveis . . . 47
3.4.2 Particionamento de hardware reconfigurável . . . 49
3.4.3 Políticas de escalonamento de configurações . . . 50
Propostas de Walder et al . . . 51
3.4.4 Relocação de configurações . . . 54

Capítulo 4: RSCM - Uma proposta de controlador de configurações 57
4.1 Estrutura do RSCM . . . 58
4.1.1 O Monitor de Reconfiguração . . . 59
4.1.2 O Controle Central de Configurações . . . 59
4.1.3 O Escalonador de Configurações . . . 61
4.1.4 A Memória de Configurações . . . 62
4.1.5 O Autoconfigurador . . . 63
4.1.6 A Interface de Configuração . . . 64
4.2 Operação do sistema RSCM . . . 65

Capítulo 5: O controlador de configurações RSCM **69**
5.1 O Monitor de Reconfiguração . . . 69
5.1.1 Validação do Monitor de Reconfiguração . . . 71
5.2 O Escalonador de Configurações . . . 72
5.2.1 Validação do Escalonador de Configurações . . . 73
5.3 O Autoconfigurador . . . 75
5.3.1 Organização dos dados na memória . . . 76
5.3.2 O Leitor de Configurações do AC . . . 77
5.3.3 O Configurador do AC . . . 78
5.3.4 Validação do Autoconfigurador . . . 79
5.4 A Interface de Configuração . . . 82
5.5 O Controle Central de Configurações . . . 83
5.5.1 O Módulo Escalonamento do CCC . . . 83
5.5.2 O Módulo Configuração do CCC . . . 85
5.5.3 Validação do Controle Central de Configurações . . . 86
5.6 A Memória de Configurações . . . 87
5.6.1 O controlador de acesso à memória externa . . . 88
5.6.2 A Interface com a Porta Serial . . . 89
5.6.3 O Multiplexador de acesso à memória de configurações . . . 90
5.7 Validação global do sistema RSCM . . . 91
5.8 Recursos de apoio à geração de controladores . . . 92
5.8.1 Parametrização do RSCM . . . 92
5.8.2 Gerando o conteúdo da Memória de Configurações . . . 93

Capítulo 6: Estudos de Caso **95**
6.1 Estudo de caso 1: Projeto LEDs . . . 95
6.1.1 Estrutura do sistema . . . 95
6.1.2 Experimentos conduzidos . . . 97
6.1.3 Resultados obtidos . . . 97
6.2 Estudo de caso 2: Projeto R8R . . . 98
6.2.1 O processador R8 . . . 99
6.2.2 O projeto R8R . . . 100
Diferenças entre o processador R8 e o processador R8R . . . 102
Modificações realizadas no controlador de configurações RSCM . . . 103
6.2.3 Experimentos conduzidos . . . 104
6.2.4 Comparação software versus hardware reconfigurável . . . 105
6.3 Análise quantitativa do sistema RSCM . . . 107

Capítulo 7: Considerações finais **111**
7.1 Resumo das contribuições . . . 111
7.2 Conclusões . . . 112

7.3 Trabalhos futuros 113

REFERÊNCIAS BIBLIOGRÁFICAS **115**

Apêndice A: Programas utilizados no projeto R8R **123**

A.1 Programa de teste do processador R8R 123
A.2 Multiplicação implementada em software 125
A.3 Divisão implementada em software 125
A.4 Raiz quadrada implementada em software 126

Lista de Tabelas

1.1 Mapeamento de um computador monoprocessado com SO multitarefa para o modelo GRS 10
1.2 Mapeamento de um processador com um conjunto dinâmico de instruções para o modelo GRS 11
1.3 Carência de suporte a RTR 13

2.1 Classificações do processo de configuração 22
2.2 Quadro comparativo entre famílias de dispositivos que habilitam RTR . . . 26

3.1 Resumo dos controladores de configurações estudados 45

4.1 Estrutura da Tabela de Alocação de Recursos (TAR) 61
4.2 Estrutura da Tabela de Dependência e Descritores (TDD) 61

5.1 Sinais manipulados no Monitor de Reconfiguração 71
5.2 Sinais manipulados no Escalonador de Configurações 73
5.3 Sinais manipulados no Leitor de Configurações do AC 78
5.4 Sinais manipulados no Configurador do AC 78
5.5 Conjunto de sinais que compõem a interface ICAP 82
5.6 Sinais manipulados no módulo de Escalonamento do CCC 84
5.7 Sinais manipulados no módulo de Configuração do CCC 86
5.8 Quadro comparativo dos controladores de acesso a memórias SRAM e SDRAM 89

6.1 Sinais manipulados da interface da R8R 101
6.2 Intruções adicionadas ao processador R8 para gerar o processador R8R . . 102
6.3 Análise quantitativa do sistema RSCM 107
6.4 Tempos de configuração de dispositivos XC2V1000 da família VirtexII . . . 108

Lista de Figuras

1.1 Estrutura genérica de um SoC 2
1.2 Estrutura do modelo GRS 6
1.3 Estados de uma configuração segundo o modelo GRS 8
1.4 Sistema PaDReH 16

2.1 Diagrama da *Cache Logic* 24
2.2 Diagrama de blocos da plataforma V2MB1000 27
2.3 Proposta de comunicação entre núcleos de [PAL02] 32
2.4 Proposta de comunicação entre núcleos de [LIM02] 33

3.1 Estrutura de um controlador de configurações proposto em [SHI98] 36
3.2 Fluxo de dados do sistema RTR RAGE [BUR97] 38
3.3 Estrutura do controlador de configurações genérico proposto em [ROB99] . 40
3.4 Estrutura do controlador de configurações implementado em [CUR03] . . . 42
3.5 Estrutura do controlador de configurações discutido em [BLO03] 44
3.6 Estrutura de software do controlador de configurações descrito em [BLO03] 45
3.7 Proposta de estrutura geral de um controlador de configurações 46
3.8 Modelos de alocação bidimensionais para dispositivos reconfiguráveis . . . 48
3.9 Modelos de alocação unidimensionais para dispositivos reconfiguráveis . . . 48
3.10 Diagrama de transição de estados de uma configuração para escalonamento com preempção proposto em [WAL03] 52
3.11 Gráfico representando o resultado do escalonamento da configuração *C7* segundo as políticas *HORIZON* e *STUFFING* apresentadas em [STE03] . . 53
3.12 O problema de colisão de configurações [COM00] 54
3.13 Sete manipulações primárias de uma configuração segundo [COM02a] . . . 55

4.1 Diagrama de blocos do controlador RSCM proposto 59
4.2 Controle Central de Configurações do sistema RSCM 60
4.3 Exemplo de grafo de escalonamento de configurações 62
4.4 Memória de Configurações do sistema RSCM 63
4.5 Autoconfigurador do sistema RSCM 64
4.6 Exemplo de operação do sistema RSCM 67

5.1 Máquina de estados do Monitor de Reconfiguração 70

5.2 Forma de onda da simulação do módulo MR 71
5.3 Máquina de estados do Escalonador de Configurações 74
5.4 Forma de onda da simulação do módulo EC 75
5.5 Organização dos dados na Memória de configurações 76
5.6 Máquina de estados do Leitor de Configurações do AC 77
5.7 Máquina de estados do Configurador do AC 79
5.8 Interface de configuração SelectMAP capturada. 80
5.9 Forma de onda da simulação do módulo AC 81
5.10 Máquina de estados do Módulo Escalonamento do CCC 84
5.11 Máquina de estados do Módulo Configuração do CCC 85
5.12 Forma de onda da simulação do módulo CCC 87
5.13 Máquina de estados do controlador de acesso à SRAM 88
5.14 Tela capturada do software de interface com a porta serial da plataforma . 90
5.15 Forma de onda da simulação do sistema RSCM completo 91

6.1 Estrutura do sistema LEDS 96
6.2 Relação tamanho do arquivo de configuração versus tempo de reconfiguração 99
6.3 Estrutura do processador reconfigurável R8R 101
6.4 RSCM adaptado para o projeto R8R 103
6.5 Nova máquina de estados do CCC adaptado para o projeto R8R. 104
6.6 Implementação do coprocessador ***multi*** em hardware x software 105
6.7 Implementação do coprocessador ***div*** em hardware x software 106
6.8 Implementação do coprocessador ***sqrt*** em hardware x software 107

Lista de Símbolos e Abreviaturas

AC	Autoconfigurador do sistema RSCM
API	*Application Programming Interface*
ASIC	*Application Specific Integrated Circuit* - Circuito Integrado de Aplicação Específica
BDP	Buffer De Palavras do Autoconfigurador do sistema RSCM
BRAM	*Block RAM* - Bloco de memória RAM em dispositivos Xilinx
CAD	*Computer Aided Design* - Projeto Auxiliado por Computador
CCC	Controle Central de Configurações do sistema RSCM
CiAE	Circuito de Aplicação Específica
CLB	*Configurable Logic Block* - Bloco Lógico Configurável
CPI	Ciclos Por Instrução
DCR	*Device Control Register* - Barramento de controle de dispositivos
DDR	*Double Data Rate*
DISC	*Dynamic Instruction Set Computer* - Computador com conjunto de instruções dinâmico
DMA	*Direct Memory Access* - Acesso Direto à Memória
DHP	*Dynamic Hardware Plugins*
DSP	*Digital Signal Processor* - Processador de Sinais Digitais

EC	Escalonador de Configurações do sistema RSCM
EDF	*Earliest Deadline First*
ESD	Entrada e Saída Dedicada
FACIN	Faculdade de Informática da PUCRS
FCFS	*First Come First Served*
FPGA	*Field Programmable Gate Array* - Matriz de Portas Lógicas Programável no Campo
GAPH	Grupo de Apoio ao Projeto de Hardware
GRS	*Generic Reconfigurable System* - Sistema Reconfigurável Genérico
HDL	*Hardware Description Language* - Linguagem de Descrição de Hardware
IC	Interface de Configuração do sistema RSCM
ICAP	*Internal Configuration Access Port* - Porta Interna de Acesso à Configuração em dispositivos Xilinx
IOB	*I/O Blocks* - Blocos de Entrada/Saída
IP	*Intellectual Property* - Propriedade Intelectual
ISP	*In-System Programmable*
JRTR	*Java Run-Time Reconfiguration*
JTAG	*Join Test Action Group*
LMB	*Local Memory Bus* - Barramento da memória local
LR	Lógica Reconfigurável
LVDS	*Low-Voltage Differential Signaling*
M	Memória

MC	Memória de Configurações do sistema RSCM
MR	Monitor de Reconfiguração do sistema RSCM
OPB	*On-chip Peripheral Bus* - Barramento de periférico interno ao circuito integrado
PaDReH	*Partial and Dynamic Reconfiguration of Hardware*
PARBIT	*PARtial BItfile Transformer*
PLB	*Processor Local Bus* - Barramento local do processador
PP	Processador Programável
PPC	*IBM PowerPC* - Processador PowerPC RISC fabricado pela IBM
PPGCC	Programa de Pós Graduação em Ciência da Computação
PROM	*Programmable Read-Only Memory* - Memória Programável Exclusivamente de Leitura
PUCRS	Pontifícia Universidade Católica do Rio Grande do Sul
RAM	*Random Access Memory* - Memória de Acesso Aleatório
RAGE	*Reconfigurable Architecture Group*
RISC	*Reduced Instruction Set Computer* - Computador com conjunto reduzido de instruções
ROM	*Read-Only Memory* - Memória Exclusiva de Leitura
RSCM	*Reconfigurable System Configuration Manager*
RTL	*Register Transfer Level* - Modelagem em nível de transferência entre registradores
RTR	*Run-Time Reconfiguration* - Reconfiguração em tempo de Execução
SJF	*Shortest Job First*

SO	Sistema Operacional
SoC	*System-on-a-Chip* - Sistema computacional integrado em um único circuito integrado
SDR	*Sistema Dinamicamente Reconfigurável*
SRAM	*Static Random Access Memory* - Memória de Acesso Aleatório Estática
SRPT	*Shortest Remaining Processing Time*
TLM	*Transaction Level Modeling* - Modelagem em nível de transação
UCF	*User Constraints File*
UCP	Unidade Central de Processamento
ULA	Unidade Lógico-Aritmética
USB	*Universal Serial Bus* - Barramento serial universal
VDI	Vetor de Descritores Internos
VHDL	*VHSIC Hardware Description Language* - Linguagem de descrição de hardware VHSIC
VHSIC	*Very High Speed Integrated Circuit* - Circuito integrado de velocidade muito alta
VLSI	*Very Large Scale Integration* - Integração em Escala Muito Alta
XDL	*Xilinx Design Language*

Capítulo 1

Introdução

Nos últimos anos vem acontecendo uma ascensão do interesse em sistemas de hardware reconfigurável e seu emprego na concepção de sistemas complexos integrados em um único circuito integrado (CI). Vários fatores impulsionam tal fenômeno. Entre eles pode-se citar a flexibilidade proporcionada por sistemas que incorporam hardware reconfigurável e os benefícios relativos à redução do tempo para o produto chegar ao mercado (em inglês, *time-to-market*). A existência de dispositivos que habilitam a implementação de sistemas dinâmica e parcialmente configuráveis aliada à carência de ferramentas para o projeto destes motiva o estudo do assunto. A semelhança entre hardware parcialmente configurável e software do ponto de vista de flexibilidade, sugere o desenvolvimento de sistemas de suporte ao hardware que desempenhem tarefas semelhantes aos de suporte a sistemas complexos de software, tais como escalonadores de módulos e sistemas operacionais. No presente trabalho apresenta-se a proposta de um controlador de configurações implementado em hardware, em contraproposta a algumas propostas anteriores, implementadas prioritariamente em software.

O presente Capítulo organiza-se da seguinte forma: Na Seção 1.1 apresentam-se algumas considerações iniciais relacionadas ao contexto do presente trabalho. A Seção 1.2 contém esclarecimentos sobre sistemas dinamicamente reconfiguráveis. Na mesma Seção apresenta-se um modelo genérico de sistema reconfigurável. Seguindo, as motivações para realização do presente estudo apresentam-se na Seção 1.3. Na Seção 1.4, trata-se a proposta de um arcabouço para projeto e implementação de sistemas dinamicamente reconfiguráveis (SDRs). A Seção 1.5 apresenta objetivos estratégicos e específicos do trabalho. No final do Capítulo, Seção 1.6, apresenta-se a organização do volume.

1.1 SoCs e Núcleos de propriedade intelectual

Em [MAR01], Martin e Chang definem o termo *System-on-Chip* (SoC) como um sistema complexo que integra a maioria dos elementos funcionais de um produto final completo em um único CI. Em geral, SoCs contém processadores programáveis (PP), memórias (M),

circuitos de aplicação específica (CiAE) de alto desempenho, interfaces com periféricos, e modernamente, também lógica reconfigurável (LR) que proporcionam uma maior flexibilidade ao sistema. De acordo com a necessidade do sistema, processadores programáveis, circuitos de aplicação específica e as áreas reconfiguráveis podem possuir ainda módulos de entrada e saída dedicados (ESD). Na Figura 1.1, apresenta-se a estrutura genérica de um SoC.

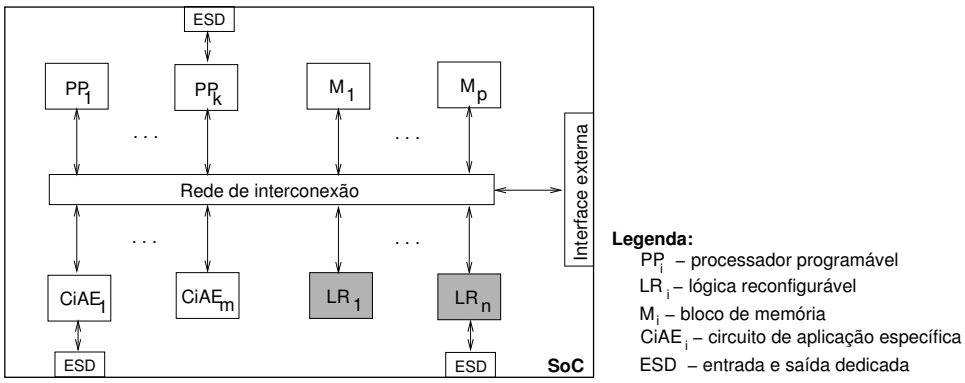

Figura 1.1: Estrutura genérica de um SoC – *Um SoC genérico é composto por processadores programáveis (PP), memórias (M), circuitos de aplicação específica (CiAE) de alto desempenho, interfaces com periféricos, lógica reconfigurável (LR) e módulos de entrada e saída dedicados (ESD).*

A tecnologia de implementação de CIs permite colocar entre várias dezenas a algumas centenas de milhões de transistores em um único dispositivo, viabilizando assim implementar SoCs. A disponibilidade de núcleos de propriedade intelectual e a disponibilidade de plataformas de prototipação rápida para validar projetos de alta complexidade tornam SoCs comercialmente viáveis do ponto de vista do projeto. Através dessas tecnologias, o projeto pode ser realizado em tempo reduzido, respeitando as exigências de tempo de chegada ao mercado para os produtos tecnológicos.

De acordo com Gupta [GUP97] e Bergamaschi [BER00], um núcleo de propriedade intelectual (em inglês *IP-Core, ou apenas núcleo*) é um módulo de hardware pré-projetado e pré-verificado que pode ser utilizado na concepção de sistemas maiores e mais complexos. Segundo Gupta [GUP97], um núcleo pode ser classificado como: soft, firm ou hard.

- *soft* - Os núcleos soft geralmente são descritos em linguagens de descrição de hardware, oferecendo flexibilidade e independência de tecnologia. Os núcleos desse tipo podem ser modificados e empregados com tecnologias de diferentes fabricantes. Os núcleos soft, em geral não são disponibilizados com garantias estritas de atendimento de características temporais;

- *firm* - Os núcleos firm são representados em níveis de abstração mais baixos, tipicamente como *netlists* que são geradas para uma dada tecnologia. Estes núcleos

podem ser parcialmente parametrizados pelo usuário projetista, permitindo que sua arquitetura seja adaptada às necessidades do projeto. Entretanto, como a *netlist* é específica para uma dada tecnologia, existem dificuldades para uso do componente na implementação de CIs junto a fabricantes diferentes;

- *hard* - Os núcleos hard são otimizados para uma tecnologia específica e não podem ser modificados pelo projetista. Estes núcleos possuem um *layout* e planta baixa pré-definidos. Possuem como maior vantagem a garantia precisa dos tempos de propagação do circuito. Sua desvantagem é a de não permitir qualquer tipo de modificação ou personalização.

SoCs integram processadores programáveis e memórias, obviamente, visando reprogramação, ou seja, alteração de seu comportamento após fabricação via modificação do software associado a esses componentes. A *lógica reconfigurável* permite que o hardware em si seja *reconfigurado*, ou seja, tenha seu comportamento alterado, de forma análoga a reprogramação de software. A reconfigurabilidade visa fornecer ao hardware flexibilidade semelhante ao software. Dessa forma, torna-se interessante integrar circuitos reconfiguráveis em SoCs porque além da flexibilidade fornecida, também o desempenho do sistema pode ser incrementado em níveis mais elevados que aqueles obtidos via alteração do software.

O presente trabalho envolve o estudo de problemas específicos relacionados a sistemas dinâmica e parcialmente reconfiguráveis. A fim de prover infra-estrutura para a operação de sistemas desta natureza, propõe-se aqui um método de controle de configurações em hardware que gerencie o processo de reconfiguração em um sistema de acordo com um cronograma de reconfigurações previamente estabelecido. A infra-estrutura aqui proposta pretende possibilitar que um sistema, composto por diversos núcleos, seja implementado em um FPGA de forma eficiente. Supõe-se que alguns destes núcleos são configurados dinamicamente no dispositivo, operação esta controlada pelo módulo de controle do processo de reconfiguração aqui proposto.

1.2 Sistemas digitais reconfiguráveis

Arranjos de portas lógicas programáveis no campo, em inglês, *Field Programmable Gate Arrays* ou simplesmente FPGAs, são dispositivos reconfiguráveis, ou seja, podem ter seu comportamento modificado, quando desejado, através da operação que configura/personaliza o dispositivos para se comportar da maneira desejada, operação esta denominada *configuração*. FPGAs foram originalmente concebidos visando a prototipação rápida de sistemas, mas atualmente sua utilização transcende em muito esta finalidade e passa a incorporar produtos finais cada vez mais freqüentemente. O uso de dispositivos reconfiguráveis permite:

- implementação de sistemas computacionais flexíveis à medida que seu comportamento pode ser alterado após sua fabricação, denominados *sistemas de computação reconfigurável*;
- implementação em uma área reduzida de um sistema conceitualmente maior que esta, através de métodos que tratam hardware reconfigurável de forma semelhante a memórias, implementando o que é denominado *hardware virtual* [DEH00].

FPGAs baseados em RAM (em inglês, *Random Access Memory*) [CHO99, CHO99a] são dispositivos naturalmente reconfiguráveis. Pode-se pensar num FPGA baseado em RAM como um hardware personalizável a cada instante pelo usuário para executar diferentes funções, através do preenchimento dos bits da memória de controle deste, em RAM. O processo de personalizar o hardware a cada instante recebe o nome de *reconfiguração*.

Desde sua introdução, FPGAs baseados em RAM [BRO92] vêm prometendo ao desenvolvimento de hardware a mesma flexibilidade que o desenvolvimento de software. Esse objetivo ainda não foi alcançado em virtude do fato dos FPGAs atuais habilitarem reconfiguração dinâmica e parcial de uma forma um tanto limitada já que a arquitetura desses dispositivos não proporciona a flexibilidade desejada. Além disso, as ferramentas de projeto e implementação de sistemas dinâmica e parcialmente reconfiguráveis são incipientes.

Em [VAH03], Vahid evidencia uma nova perspectiva onde o hardware torna-se tão flexível quanto o software, principalmente devido ao emprego de lógica reconfigurável e também ao uso de linguagens de alto nível para criar/desenvolver hardware. O advento e a vertiginosa ascensão de complexidade e uso de dispositivos de hardware reconfigurável geram a necessidade de novos paradigmas de projeto em todos os níveis de abstração, desde os inferiores aos mais abstratos [ZHA00]. *Sistemas reconfiguráveis* são aqueles onde o hardware pode, após sua fabricação, ter alterada, ainda que parcialmente, sua funcionalidade/comportamento.

Maestre et al. [MAE01, MAE01a] citam que a *computação reconfigurável*, sistemas computacionais que empregam lógica reconfigurável, consolida-se cada vez mais como uma alternativa de projeto viável para implementar aplicações de computação intensiva, principalmente no mercado de Processador de Sinais Digitais (em inglês, *Digital Signal Processor* ou DSPs) e aplicações de multimídia. A computação reconfigurável é uma alternativa para projeto de aplicações que envolvem redes neurais [POR02], sistemas móveis [SMI02], ou seja, sistemas de diversas naturezas.

Alguns dispositivos podem ser reconfigurados apenas *totalmente*, dessa forma, todos os itens configuráveis do dispositivo devem receber uma configuração antes deste poder ser utilizado. Outros, por sua vez, podem ser reconfigurados *parcialmente*, onde alguns dos bits de configuração podem ser alterados de cada vez. Estes são chamados *dispositivos parcialmente reconfiguráveis*. O estudo aqui apresentado relaciona-se especificamente a dispositivos dessa natureza.

Como classificou Sanchez [SAN99], reconfigurações podem ser dinâmicas ou estáticas. Se o sistema não necessita ter seu processamento interrompido (sem disrupção) para que uma reconfiguração seja realizada então ele é dito *dinâmico*, caso contrário, é dito *estático*.

Pode-se ainda classificar dispositivos reconfiguráveis de acordo com o *tamanho do grão configurável*. Entende-se por *grão* a menor unidade configurável da qual um dispositivo é composto. Modernamente, se as configurações se dão no nível de portas lógicas ou funções Booleanas de até quatro entradas diz-se que o dispositivo possui *grão pequeno*. Se estas se dão sobre unidades maiores, tais como ULAs, diz-se que o dispositivo possui *grão médio*. Quando estas se dão em unidades ainda maiores, tais como um microprocessador, diz-se que o dispositivo é de *grão grande*.

Um conceito importante a ser tratado em SDRs é o de *densidade funcional*, que representa a taxa de utilização dos recursos de um dispositivo reconfigurável, ou seja, durante a execução do sistema, qual o nível de utilização de seus recursos. O conceito de densidade funcional é apresentado em [WIR97] como o inverso do produto entre a área ocupada pelo circuito e o tempo de execução do mesmo. A Equação 1.1 apresenta a relação entre densidade funcional (D) e o produto da área ocupada pelo circuito (A) pelo tempo de execução (T). De acordo com essa Equação, se dois circuitos ocupam a mesma área mas gastam tempos diferentes para executar, então aquele que gasta mais tempo terá uma densidade funcional menor. Da mesma forma, se dois circuitos gastam o mesmo tempo para executar mas possuem áreas diferentes, aquele com maior área possui menor densidade funcional. Conceitos similares aos apresentados por Wirthlin [WIR97] foram desenvolvidos de forma independente por DeHon [DEH98]. Wirthlin [WIR97] apresenta como um dos principais fatores que devem ser analisados para avaliar o uso de sistemas reconfiguráveis o *tempo de configuração* (ou reconfiguração).

$$D = \frac{1}{A \times T} \tag{1.1}$$

O presente trabalho envolve o estudo de problemas específicos relacionados a sistemas dinâmica e parcialmente reconfiguráveis. A fim de prover suporte ao desenvolvimento e operação de sistemas desta natureza, propõe-se aqui, um método de controle de configurações em hardware para gerenciar os processos de configurações em um sistema de acordo com um escalonamento conhecido. Desta forma, pretende-se possibilitar que um FPGA possa ser utilizado de maneira eficiente por diversos núcleos, sejam eles relacionados entre si ou não. Tais núcleos são configurados dinamicamente no dispositivo, operação esta controlada através do método proposto.

1.2.1 Modelo genérico para sistemas reconfiguráveis

Em Seções anteriores, o conceito de sistema reconfigurável foi apresentado de uma maneira um tanto restrita. Mais especificamente, teve-se em mente dispositivos de hardware,

em particular FPGAs baseados em RAM. Na realidade, pode-se generalizar o conceito para abranger vários sistemas de naturezas distintas como sistemas reconfiguráveis. Coloca-se a questão de qual seria uma boa definição de sistema reconfigurável. A Figura 1.2 ilustra uma proposta de modelo genérico para representar os fluxos de informação de controle em sistemas reconfiguráveis.

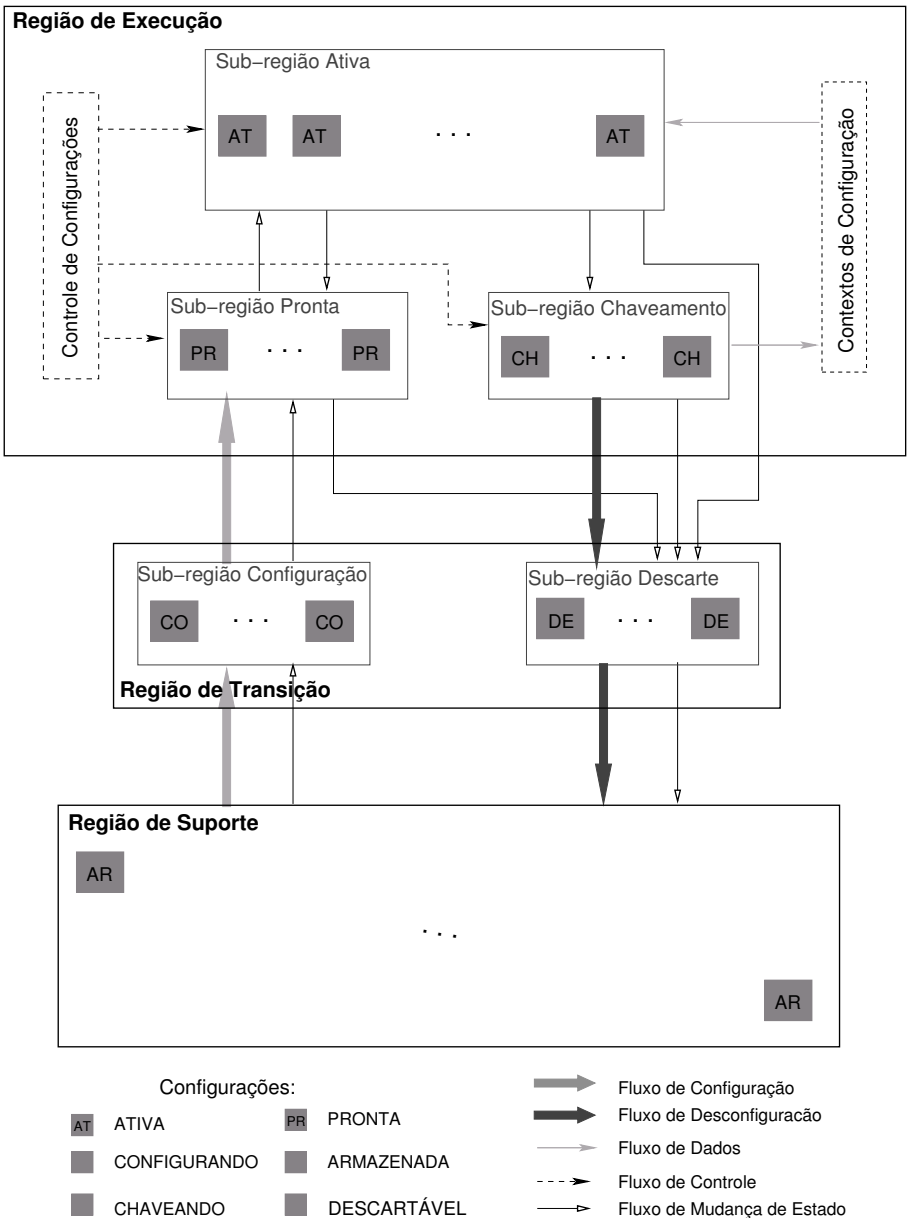

Figura 1.2: Estrutura do modelo GRS – *O modelo Generic Reconfigurable System é utilizado para representar sistemas reconfiguráveis. Dependendo do nível de abstração em que se observa o sistema, ele pode ser considerado como reconfigurável ou não.*

A Figura 1.2 ilustra o modelo proposto para representar sistemas reconfiguráveis de forma genérica, denominado *Generic Reconfigurable System* (GRS). Na Figura, nota-se que um sistema reconfigurável é composto de três regiões: região de suporte, região de transição e região de execução. Em todas as regiões ilustradas são encontradas *configurações*. Neste contexto, *configuração* consiste de um conjunto de informações que pode ser usado para personalizar o comportamento desempenhado por uma parte de um sistema reconfigurável em um dado instante.

A *Região de Suporte* é o repositório de configurações. Tipicamente, todas as configurações que um sistema reconfigurável necessita em algum momento estão presentes nesta região. A *Região de Transição*, como o próprio nome indica é um repositório onde residem temporariamente configurações em trânsito entre as regiões de execução e de suporte. Esta região subdivide-se em duas sub-regiões. A primeira delas é a *Sub-região de Configuração*, na qual encontram-se as configurações que estão sendo inseridas na região de execução. A segunda é a *Sub-região de Descarte*, que abrange as configurações em processo de desconfiguração ou que estão prontas para serem desconfiguradas, uma vez que já tiveram seu contexto salvo e não mais são necessárias no sistema em curto prazo.

A *Região de Execução* corresponde à parte efetivamente operacional do sistema reconfigurável, aquela capaz de executar trabalho útil. Esta região está subdividida em cinco sub-regiões, sendo duas fixas e três de natureza reconfigurável. As sub-regiões fixas servem ao controle do processo de reconfiguração do sistema. São elas a sub-região de controle de configurações e sub-região de contextos de configuração. As três sub-regiões reconfiguráveis são: sub-região ativa, sub-região pronta e sub-região de chaveamento. A *Sub-região de Contextos de Configuração* armazena dados dinamicamente gerados por configurações que necessitam ser comunicados a outras configurações, após sua desconfiguração. A *Sub-região de Controle de Configurações* controla o processo de inserção, remoção e movimentação de configurações entre as sub-regiões ativa, de chaveamento e pronta. A *Sub-região Ativa* engloba as configurações que produzem trabalho útil no sistema reconfigurável. A *Sub-região Pronta* engloba as configurações prontas para passarem à sub-região ativa e que aguardam sinalização para tanto da região de controle de configurações migrando então para a sub-região ativa. A *Sub-região de Chaveamento* agrupa as configurações que se encontram salvando contexto para comunicação com outras configurações. Desta sub-região as configurações apenas podem passar para as sub-regiões pronta ou de descarte.

Como pode ser visto na legenda apresentada na Figura 1.2, uma configuração no modelo GRS está sempre em um dos seis estados distintos possíveis. Na Figura 1.3, apresenta-se um diagrama de transição de estados que mostra a forma do fluxo de controle de configurações em qualquer sistema reconfigurável.

Uma configuração é dita *armazenada* quando existe dela apenas uma cópia, a qual está na região de suporte. Na região de transição, as configurações são encontradas em dois estados: configurando e descartável. Como o próprio nome indica, uma configuração

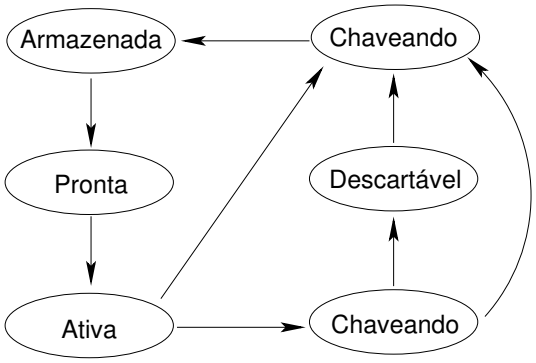

Figura 1.3: Estados de uma configuração segundo o modelo GRS – *Uma configuração pode estar em um de seis estados, são eles: armazenada, configurando, pronta, ativa, chaveando e descartável.*

configurando é aquela que está sendo configurada, e portando armazenada na sub-região configuração. Uma configuração *descartável* é aquela que já salvou seu contexto e está pronta para retornar à região de suporte. Dentro da região de execução são encontradas configurações em três estados: prontas, ativas ou chaveando. Uma configuração *pronta*, está configurada e aguarda na sub-região pronta até que possa se tornar *ativa* (na sub-região execução) ou passar ao estado descartável (quando irá para região de descarte). Quando uma configuração pára de executar, ou seja, deixa de ser ativa, ela ou passa para o estado *chaveando*, caso seja necessário salvar seu contexto para comunicação, ou passa diretamente para o estado descartável. Configurações no estado chaveando são mantidas na sub-região de chaveamento.

Cabe ainda, definir os processos que as configurações sofrem durante a operação do sistema. O *processo de configuração* consiste em personalizar o comportamento desempenhado por uma parte de um sistema reconfigurável em um dado instante. Por outro lado, o *processo de reconfiguração* possui o mesmo objetivo e consiste no ato de sobrescrever uma antiga configuração, ou seja, "repersonalizar" o sistema. O *processo de desconfiguração* consiste em permitir que uma configuração seja desfeita ou até mesmo "despersonalizar o sistema".

A seguir apresentam-se três exemplos de sistemas computacionais e considerações a respeito dos mesmos que os classificam como sistema reconfigurável, nos dois primeiros casos, e um sistema não reconfigurável no último caso.

Um computador como um sistema reconfigurável

Em um computador monoprocessado com sistema operacional multitarefa os programas (configurações armazenadas) são estocados em uma área de armazenamento secundário

(região de suporte). Quando um destes programas é carregado na memória virtual pelo Sistema Operacional (SO) para ser executado, ela passa a ser um programa em processo de carga (configurando). Logo após ser carregado este adquire área em memória e passa ser um processo pronto para execução (uma configuração pronta) aguardando até o instante no qual o escalonador (parte da sub-região de controle de configuração) o selecionar para execução, quando se torna um processo em execução (configuração ativa). Num dado instante, esse processo em execução será preemptado pelo escalonador, ou simplesmente terminará sua tarefa, e neste instante o processo (configuração chaveando) poderá necessitar salvar contexto para comunicação com outros processos que irão executar, através de pipes, sockets ou outro mecanismo. Quando um processo não é mais usado pelo sistema, ele deve retornar a região de armazenamento. Então, passa a ser um programa em operação de descarga (configuração descartável) da memória e pode nesse instante salvar algumas informações sobre seu próprio contexto. Isto é realizado, pois na próxima vez que executar o programa pode carregar consigo estas informações, que refletem o estado no qual seu contexto foi salvo na execução anterior.

O mapeamento conceitual de um computador monoprocessado com um sistema operacional multitarefa para o modelo GRS encontra-se resumido na Tabela 1.1.

Com base nas observações expostas acima pode-se concluir que um computador pessoal pode ser considerado um sistema reconfigurável de acordo o modelo GRS.

Um DISC como um sistema reconfigurável

Alguns processadores contêm um conjunto de instruções dinâmico em contraproposta aos processadores convencionais, os quais possuem um conjunto de instruções fixo. Wirthlin e Hutchings tratam esse tipo de processador em [WIR95, WIR95a], denominado *Dynamic Instruction Set Computer* ou DISC. Os processadores dessa natureza valem-se de técnicas de reconfiguração parcial para transformar seu conjunto de instruções. Cada uma das instruções do DISC é implementada na forma de um circuito independente (configuração). As instruções são configuradas sob demanda, de acordo com a execução do programa. Um controlador de configurações controla o fluxo de configurações de instruções; uma memória armazena os arquivos de configuração das instruções e um computador hospedeiro encarrega-se de enviar os dados de configuração para o sistema. Um módulo fixo do sistema, denominado controlador global, dedica-se a controlar os contextos das instruções.

O mapeamento conceitual de um processador com um conjunto dinâmico de instruções para o modelo GRS encontra-se resumido na Tabela 1.2.

Com base nas observações expostas em tal Tabela pode-se concluir que o processador com um conjunto dinâmico de instruções pode ser considerado um sistema reconfigurável de acordo o modelo GRS.

Tabela 1.1: Mapeamento de um computador monoprocessado com SO multitarefa para o modelo GRS – *O computador monoprocessado com SO multitarefa é um sistema reconfigurável de acordo com as características destacadas no modelo GRS.*

Elemento do modelo RSG	***Interpretação de um computador monoprocessado com SO multitarefa***
Região de Suporte	armazenamento secundário (e.g. em disco rígido)
Região de Execução	processos em execução + kernel do SO
Região de Transição	parte da memória virtual que armazena os programas em processo de carga ou salvando contexto
Sub-região Ativa	processo ativo (no caso desse computador monoprocessado só existe um processo ativo a cada instante)
Sub-região Pronta	parte da memória virtual de armazena os processos prontos para execução
Sub-região de Chaveamento	parte da memória virtual que armazena os programas salvando informações para serem comunicadas a outros processos após seu término
Sub-região de Configuração	parte da memória virtual que armazena os programas em processo de carga
Sub-região de Descarte	programas salvando sua configuração interna
Sub-região de Controle de Configuração	carregador (loader) + parte do kernel SO que controla escalonamento de processos, preempção e comunicação entre processos
Sub-região Contexto de Configuração	área de memória virtual usada para comunicação entre os processos tais como pipes, sockets
Configuração	código executável de um programa
Configuração Armazenada	programa em meio de armazenamento secundário
Configuração Ativa	programa atualmente de posse da UCP, executando
Configuração Configurando	programas em processo de carga na memória virtual
Configuração Pronta	processos prontos
Configuração Descartável	programa salvando seu contexto interno para próxima execução
Configuração Chaveando	programas salvando informações para serem comunicadas a outros processos após seu término

Um contra-exemplo

Um sistema embarcado com hardware fixo e software monotarefa fixo não pode ser modelado como um sistema reconfigurável tal como o proposto na Figura 1.2. Esse sistema não contém sequer uma região que possa ser dita reconfigurável. Toda essência do modelo GRS reside na região de execução, na qual os sistemas são (re)configurados. O sistema embarcado com hardware fixo e software monotarefa fixo é, portanto um contra-exemplo de um sistema reconfigurável.

Dependendo do nível de abstração em que se observa o sistema, o mesmo pode ser considerado como reconfigurável ou não. Pode-se melhor esclarecer esse fenômeno através do seguinte exemplo prático: atualmente nos caixas de supermercado utilizam-se computadores pessoais para registrar compras e emitir notas fiscais. Do ponto de vista do usuário do sistema (i.e. o caixa), o computador comporta-se apenas como uma máquina registradora sem qualquer característica relacionada à reconfiguração. Mas, do ponto de vista do

Tabela 1.2: Mapeamento de um processador com um conjunto dinâmico de instruções para o modelo GRS – *O processador com um conjunto dinâmico de instruções é um sistema reconfigurável de acordo com as características destacadas no modelo GRS.*

Elemento do modelo RSG	***Interpretação em um processador com um conjunto dinâmico de instruções***
Região de Suporte	memória que armazena as instruções
Região de Execução	parte do dispositivo onde encontram-se as instruções configuradas
Região de Transição	parte do dispositivo onde encontra-se as instruções sendo configuradas ou que já foram utilizadas
Sub-região Ativa	parte do dispositivo onde encontra-se a instrução em execução
Sub-região Pronta	parte do dispositivo onde encontram-se instruções configuradas com exceção da instrução em execução
Sub-região de Chaveamento	inexistente
Sub-região de Configuração	parte do dispositivo onde encontra-se uma configuração que está sendo configurada
Sub-região de Descarte	parte do dispositivo onde encontra-se uma configuração que já foi executada
Sub-região de Controle de Configuração	parte do dispositivo que dedica-se a controlar a configuração de instruções, no caso do DISC, composta por: computador hospedeiro e Controle Global
Sub-região Contexto de Configuração	inexistente
Configuração	instrução
Configuração Armazenada	instrução na memória de instruções
Configuração Ativa	instrução configurada no dispositivo que está executando
Configuração Configurando	instrução sendo configurada no dispositivo
Configuração Pronta	instrução configurada no dispositivo mas que não se encontra em execução ainda
Configuração Descartável	instrução que terminou sua execução mas que ainda encontra-se configurada no dispositivo
Configuração Chaveando	inexistente

responsável pelo suporte técnico do sistema, o computador não é apenas uma máquina registradora e, além disso, suporta a execução de diversos programas, tais como sistema operacional, controladores de impressora etc. Tal informação é transparente ao usuário pois este adota um nível diferente de abstração, nesse caso um nível mais alto. Como a cada instante um programa diferente é executado, de acordo com o nível de abstração adotado pelo supervisor, o sistema do supermercado pode ser classificado como reconfigurável.

1.3 Motivações

Hoje, pode-se notar o crescimento acentuado do interesse em computação configurável, como evidenciado por Zhang e Ng [ZHA00]. A flexibilidade proporcionada pela computação reconfigurável é um fator relevante que pode aumentar a janela de tempo na qual o produto permanece no mercado (tempo de vida do produto). Como no caso dos sistemas de software, que recebem atualizações constantes, o hardware implementado em disposi-

tivos reconfiguráveis também pode usufruir desta estratégia para se manter útil por mais tempo.

O ciclo de desenvolvimento de sistemas vem diminuindo em duração por pressões de mercado. O uso da tecnologia configurável e o reuso de núcleos de propriedade intelectual tornam o projeto de SoCs mais rápido, adequando-o às restrições do mercado. O tempo para o produto chegar ao mercado pode ser reduzido de maneira significativa.

Um dos fatores mais atraentes que podem tornar o uso de computação reconfigurável uma alternativa interessante é a possibilidade de implementar em uma área reduzida um sistema conceitualmente maior que esta, implementando *hardware virtual* [DEH00]. A principal vantagem da virtualização do hardware é a redução do custo do produto final. Como a relação custo/portas equivalentes não é linear para plataformas de implementação comerciais, ou seja, uma plataforma com dez milhões de portas lógicas não custa exatamente dez vezes o custo de uma com um milhão de portas, na verdade custa muito mais. Em virtude disso, muitas vezes torna-se inviável implementar um sistema utilizando plataformas de grande porte. Nesse caso, duas alternativas podem ser tomadas: usar técnica de hardware virtual em plataformas de menor porte ou agrupar diversas plataformas de menor porte para desenvolver o sistema.

O uso de técnicas de reconfiguração em tempo de execução (em inglês, *Run-Time Reconfiguration* ou RTR) pode resultar em uma economia de recursos e um melhor aproveitamento da área, já que partes do sistema não utilizadas num determinado instante podem ser "removidas" do hardware físico para dar lugar a uma outra parte do sistema necessária no momento. Desta forma, o emprego de RTR pode melhorar a densidade funcional do sistema. Um exemplo de sistemas que podem valer-se das características de RTR para alcançar uma maior difusão comercial diz respeito a detecção de intrusão em redes de computadores. Esta tarefa necessita que uma grande quantia processamento de dados seja realizada utilizando um conjunto de regras. Esse conjunto pode ser alterado em função do padrão de tráfico instantâneo da rede. A rápida troca dessas regras bem como o alto desempenho necessário inviabilizam soluções puramente implementadas em software.

Alguns pesquisadores demonstraram vantagens do emprego de técnicas de reconfiguração parcial. Hauck, em [HAU98], apresenta computação reconfigurável como o verdadeiro paradigma de propósito geral, já que a flexibilidade não é mais representada apenas pela reprogramação do software mas também pela reconfiguração do hardware, ou seja, o sistema como um todo é de propósito geral. Shirazi, em [SHI98], estima que o uso de técnicas RTR é capaz de superar as reduzidas capacidade e desempenho de sistemas configuráveis comparados a ASICs. Conforme Vissers [VIS03], a análise e a exploração de RTR é o maior passo para plataformas de computação reconfigurável, permitindo atender melhor a compromissos entre espaço e tempo.

RTR carece de suporte como evidenciado por Zhang e Ng em [ZHA00]. Neste contex-

to, o suporte refere-se principalmente a ferramentas habilitadoras de uso da tecnologia e infra-estrutura de implementação SDRs. Existe uma carência de ferramentas de projeto e verificação de SDRs sob a forma de ferramentas de projeto auxiliado por computador (CAD, em inglês *Computer Aided Design*). Outra carência refere-se a infra-estrutura necessária para a operação de sistemas desta natureza. Esta infra-estrutura compreende:

- dispositivos de hardware (i.e. plataformas de prototipação) que permitam RTR;
- núcleos específicos para o controle da operação do sistema, problema este atacado no presente trabalho;
- interfaces padronizadas entre os núcleos que compõem o sistema [OST03, BRI03].

Para criar SDRs necessita-se sobretudo de ferramentas de projeto (e.g. geração automatizada de configurações, de interfaces de comunicação entre partes fixas e reconfiguráveis de SDRs etc) e ferramentas de verificação (e.g. simulação da dinâmica de reconfiguração em SDRs etc). Na Tabela 1.3 apresenta-se um resumo das carências de suporte a SDRs. Como citado, o presente trabalho objetiva atacar uma das carências relacionadas à infra-estrutura de reconfiguração para SDRs. Outros trabalhos paralelos [OST03, BRI03] visam atacar as demais carências.

Tabela 1.3: Carência de suporte a RTR – *As carências compõem-se de ferramentas e infra-estrutura necessária para a implementação e operação de tais sistemas.*

Suporte	Ferramentas	- Projeto de SDRs - Verificação de SDRs
	Infra-estrutura	- Dispositivos habilitadores de SDRs - **Núcleos para controlar a operação de SDRs** - Interfaces padronizadas entre núcleos

A implementação de SDRs pressupõe uma infra-estrutura composta por núcleos específicos para controle e operação de SDRs, como citado. Dentre estes núcleos destaca-se aquele responsável por gerenciar o processo de configuração, ou seja, controlar qual configuração deve ser inserida no hardware físico e qual deve ser "removida". Esse núcleo realiza tarefas similares às executadas pela parte de um sistema operacional em computadores, responsável por gerenciar memória virtual e carregar processos para executar no processador, seguindo os comandos passados por um escalonador de processos [TAN01].

Desenvolver pesquisas nesta área é importante na medida que ela se apresenta promissora para aumentar a eficiência global de sistemas e ainda apresenta muitas carências, ou seja, é um campo aberto para o desenvolvimento de trabalhos. As carências maiores são a falta de suporte adequado, sobretudo ferramentas de CAD e plataformas para dar suporte a sistemas dinâmica e parcialmente reconfiguráveis. Já que a questão do suporte de dispositivos é uma tarefa que requer maior investimento e não pode facilmente ser aplicada no

contexto institucional em que se realiza este trabalho, propõe-se aqui atacar o suporte de métodos e ferramentas para sistemas dinâmica e parcialmente reconfiguráveis.

Mais especificamente, a motivação para este trabalho é o desenvolvimento de um controlador de configurações em hardware. Como poder-se-á notar no Capítulo 3 existem poucas implementações de controladores de configurações e alguns dos assuntos relacionados às alternativas de implementação ainda não foram tratadas de maneira detalhada. A maioria dos modelos propostos não passa de conceitos que jamais foram implementados e, portanto não abordam as dificuldades enfrentadas na realidade de SDRs, já que as restrições quanto ao suporte não são consideradas, tal como a arquitetura de FPGAs comerciais.

1.4 PaDReH - arcabouço para SDRs

A presente Seção contextualiza o trabalho aqui proposto. O objetivo é situar o leitor com relação onde se posiciona o sistema proposto dentro do fluxo que representa o desenvolvimento e implementação de sistemas dinamicamente reconfiguráveis.

O presente trabalho é parte de um ambiente para desenvolvimento e implementação de SDRs denominado *Partial and Dynamic Reconfiguration of Hardware* (PaDReH), cujo objetivo é permitir aos desenvolvedores gerarem um sistema complexo, conceitualmente maior que o dispositivo alvo com garantia de que o mesmo se comportará da maneira adequada. Pretende-se com este ambiente obter vantagens com a utilização de técnicas de reconfiguração dinâmica e parcial de hardware. O sistema PaDReH divide-se em três partes:

- **Módulo de captura e validação funcional de projeto**. Este módulo responsabiliza-se pela descrição e validação do sistema no nível transacional (TLM, em inglês *Transaction Level Modeling*) e tradução para o nível de transferência entre registradores (RTL, em inglês *Register Transfer Level*). A descrição no nível RTL é a entrada do módulo de *particionamento e escalonamento*. Este módulo particiona espacialmente o sistema de acordo com informações tais como o comportamento do sistema e características do SoC alvo da implementação. Trabalhos relativos a esse módulo são apresentados por Moreno, em [MOR03].

- **Módulo de particionamento e escalonamento**. Este módulo gera arquivos que descrevem o comportamento do sistema na forma de uma descrição em linguagem de descrição de hardware (em inglês, *Hardware Description Language* ou HDL) para o módulo de *síntese física e infra-estrutura de reconfiguração*. Trabalhos relativos a esse módulo são apresentados por Marcon, em [MAR02].

- **Módulo de síntese física e infra-estrutura de reconfiguração**. Módulo que gera configurações totais e/ou parciais de acordo com o particionamento, adiciona o módulo controlador parametrizado conforme as características do sistema, e gera a

implementação física de uma rede de interconexão entre os núcleos. O submódulo de *controle reconfiguração* gerencia a operação do sistema em *tempo de execução*. Ele recebe como entradas: um grafo contendo a definição das relações de dependência entre os núcleos do sistema, fornecido pelo módulo de *particionamento e escalonamento*, e arquivos de configuração total(ais) e parciais gerados por parte do módulo de *síntese física e infra-estrutura de reconfiguração*. Trabalhos relativos a esse módulo são apresentados por Brião, em [BRI03] e Ost, em [OST03].

O submódulo de controle reconfiguração, proposto nesse trabalho, faz parte do módulo de síntese física e infra-estrutura de reconfiguração como pode ser visto na Figura 1.4. De acordo com o fluxo supra descrito, percebe-se que os dois primeiros módulos fazem parte da fase de projeto do SDR enquanto que o último, além das parametrizações realizadas na fase de projeto, atua também na execução/implementação do mesmo.

O desenvolvimento do arcabouço proposto implica abordar temas tais como:

- o particionamento de hardware em partes fixas e reconfiguráveis;
- o escalonamento de configurações para execução;
- a forma de comunicação entre configurações;
- a geração de arquivos de configuração parciais e totais;
- a configuração de dispositivos parcialmente reconfiguráveis;
- a representação de informações a respeito do cronograma de escalonamento;
- o gerenciamento da operação de configurações.

Este trabalho trata apenas do desenvolvimento do submódulo de *controlador de reconfiguração*. Assume-se, portanto, que todas as partes referentes aos outros subsistemas encontram-se operacionais, quais sejam os arquivos de configuração, o cronograma de escalonamento e, a estrutura de comunicação entre os módulos.

1.5 Objetivos

Os principais objetivos desse trabalho são desenvolver um controlador de configurações em hardware que servirá como suporte ao desenvolvimento de sistemas que se valem de estratégias RTR e contribuir para sanar algumas das carências identificadas para implementação e uso de SDRs.

Os objetivos estratégicos do trabalho proposto são:

1. Dominar a tecnologia de SDRs sobre FPGAs comerciais;
2. Contribuir para a redução das carências de infra-estrutura fornecida à operação de SDRs.

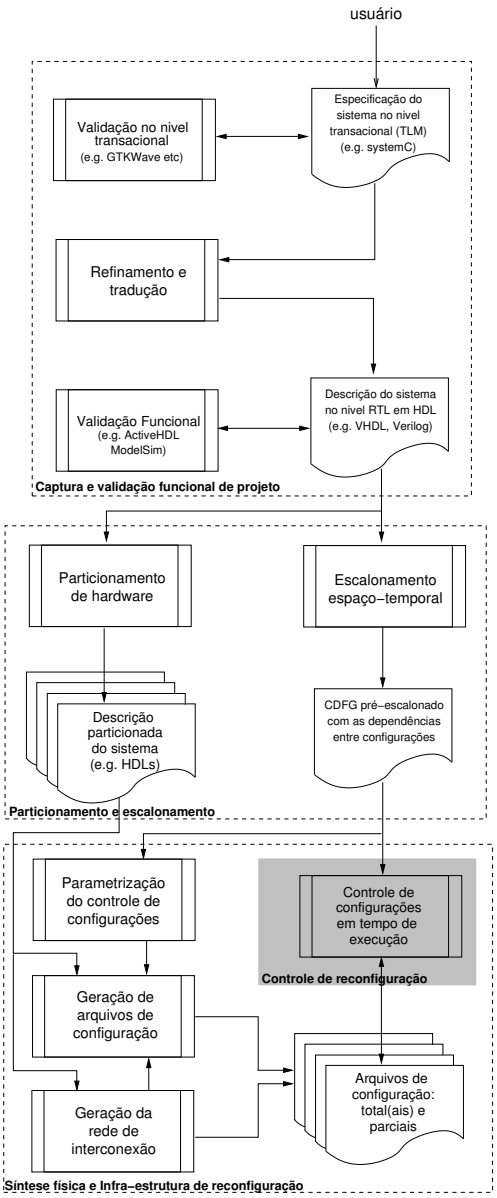

Figura 1.4: Sistema PaDReH – *Fluxo de desenvolvimento e implementação de SDRs de acordo com o arcabouço PaDReH proposto.*

Como objetivos específicos do trabalho proposto apresentam-se:

1. Desenvolver um hardware autoconfigurador para dispositivos configuráveis dinâmica e parcialmente e prototipá-lo;
2. Definir a estrutura geral do ambiente para controlar a reconfiguração de núcleos de propriedade intelectual em sistemas reconfiguráveis;

3. Validar a parte implementada do sistema definido, usando uma aplicação alvo selecionada;
4. Desenvolver uma estratégia de controle do processo de reconfiguração dinâmica e parcial em uma plataforma de prototipação comercial.

1.6 Organização do volume

O restante deste documento encontra-se distribuído em 6 Capítulos descritos a seguir.

No Capítulo 2 apresentam-se informações relacionadas ao estado da arte em pesquisas na área de computação reconfigurável, incluindo um estudo abrangendo as tecnologias habilitadoras e ferramentas de apoio ao projeto RTR.

A seguir, no Capítulo 3 discute-se o resultado de pesquisas relevantes relacionadas ao controle de configurações seguido pela discussão de modelos e implementações de controladores de configurações propostos na literatura. Informações sobre estratégias para alocação de recursos reconfiguráveis, estratégias de particionamento de hardware e políticas de escalonamento de configurações também são tratadas. Ao final desse Capítulo, apresenta-se um resumo das principais características de um controlador de configurações que servirá de base ao modelo aqui proposto.

O Capítulo 4 contém a proposta de implementação de um controlador de configurações denominado *Reconfigurable System Configuration Manager* (RSCM). Nesse Capítulo, apresenta-se detalhadamente cada um dos módulos constituintes do modelo.

No Capítulo 5 descreve-se em detalhe a implementação de cada um dos módulos apresentados no Capítulo 4. São tratados os passos de projeto, verificação e prototipação de cada um deles. Ao final desse Capítulo, discute-se as etapas de validação e prototipação do sistema unificado, módulo a módulo.

O Capítulo 6 tabula os resultados do desenvolvimento do sistema. Além disso, propõe-se dois estudos de caso a serem empregados com objetivo de comprovar a validade da presente proposta.

O último Capítulo apresenta um resumo das contribuições deste trabalho, algumas conclusões e um certo número de propostas de trabalhos futuros.

Capítulo 2

Reconfiguração dinâmica e parcial

Nos últimos anos, diversos trabalhos de pesquisa em reconfiguração dinâmica e parcial têm sido desenvolvidos, tais como os descritos em [HOR02] [LIM02] [SAN99] [MES02]. Estes apresentam soluções para problemas em diferentes áreas valendo-se dos benefícios da computação reconfigurável. Exemplos das áreas endereçadas são o projeto de roteadores em redes de computadores e o processamento de áudio e vídeo. Um conjunto distinto de trabalhos propõe métodos e ferramentas de suporte à reconfiguração parcial. Exemplo destes últimos são as propostas de controladores do processo de configuração [SHI98] [BUR97] [ROB99] [CUR03].

Para a implementação de qualquer sistema computacional complexo, seja ele dinâmica e parcialmente reconfigurável ou não, faz-se necessário suporte de hardware e software. No caso de sistemas RTR, o suporte de hardware é representado por plataformas de prototipação entre outros dispositivos enquanto o software é representado por ferramentas de projeto. Como citado anteriormente, as ferramentas de suporte ao desenvolvimento de sistemas dinamicamente reconfiguráveis (SDRs) são poucas. Além disso, as existentes não satisfazem as necessidades reais dos projetistas de SDRs, dentre elas a automatização no projeto de sistemas dessa natureza para que os mesmos possam respeitar às restrições impostas pelo mercado com relação ao tempo para o produto chegar ao mercado.

Na primeira seção do presente Capítulo apresenta-se uma proposta de classificação para o processo de configuração. Na Seção 2.2 apresenta-se considerações relacionadas ao suporte à reconfiguração dinâmica e parcial evidenciando tecnologias habilitadoras, ferramentas de software para geração e manipulação de configurações parciais, e estratégias de interconexão de núcleos que permitem a reconfiguração dos mesmos durante a execução do sistema.

2.1 Classificação do processo de configuração

Muitas vezes referencia-se o termo configuração, o processo de transferir dados de uma memória de configurações para o dispositivo de forma a personalizá-lo/configurá-lo. Visando caracterizar os parâmetros que influenciam o processo de configuração, propõe-se aqui seis critérios para classificá-lo, listados abaixo e discutidos a seguir, nas próximas seis Subseções.

1. Largura do caminho de dados de configuração;
2. Comando do processo de configuração;
3. Localização da memória de configuração;
4. Tamanho do arquivo de configuração;
5. Dimensão dos módulos manipulados na configuração;
6. Formato do arquivo de configuração.

2.1.1 Largura do caminho de dados de configuração

De acordo com a largura do barramento através do qual os dados de configuração são comunicados ao sistema reconfigurável, pode-se classificar uma dada configuração em configuração serial ou paralela. Em uma *configuração serial*, os dados de configuração são enviados para o dispositivo bit a bit. Em uma *configuração paralela* a configuração é realizada através do envio de n bits de dados de configuração de cada vez ao dispositivo ($n > 1$). No caso especifico dos dispositivos da família Virtex comercializados pela Xilinx, a configuração pode ser realizada serialmente ou ainda paralelamente com uma largura de 8 bits, modo SelectMAP segundo denominação do fabricante.

2.1.2 Comando do processo de configuração

Uma outra forma de classificar uma configuração é de acordo com o dispositivo que comanda o processo de configuração, ou seja, aquele que gera o sinal de relógio para o processo de configuração. Também aqui existem duas classes: dependente e autônoma. No caso de uma *configuração dependente* o relógio de configuração é gerado por um dispositivo externo ao sistema/dispositivo reconfigurável. Na *configuração autônoma*, o relógio de configuração é gerado pelo próprio sistema/dispositivo que está sendo configurado. É responsabilidade de dispositivos externos apenas armazenar as configurações e fornecer os dados de configuração segundo a taxa determinada pelo dispositivo.

2.1.3 Localização da memória de configuração

De acordo com a localização da memória de configuração em relação ao sistema configurável, pode-se classificar o processo de configuração em: com memória externa ou com memória interna. A primeira delas dita *configuração com memória externa*, ocorre quando o arquivo de configuração está armazenado fora do sistema/dispositivo reconfigurável. Um

exemplo da primeira classificação ocorre quando o sistema é uma plataforma de prototipação que contém um FPGA baseado em RAM, e o processo de (re)configuração se dá a partir de um computador hospedeiro conectado a plataforma via um cabo de programação. Neste caso, as configurações são arquivos armazenados no subsistema de memória secundária do hospedeiro. A outra classificação, *configuração com memória interna*, aplica-se quando o arquivo de configuração está armazenado em algum dispositivo (e.g. flash, RAM, disco etc) do sistema reconfigurável. Neste caso, são usados recursos internos ao sistema para a transmissão de dados de configuração ao dispositivo.

2.1.4 Tamanho do arquivo de configuração

O processo de configuração também pode ser classificado de acordo com o tamanho relativo do arquivo de configuração utilizado. As duas classes nas quais serão divididas as configurações são: configuração total e configuração parcial. Quando o arquivo de configuração possuir as informações necessárias para configurar todo dispositivo/sistema a configuração é dita *total*. Se as informações forem suficientes para configurar apenas parte do dispositivo/sistema a configuração é dita *parcial*.

2.1.5 Dimensão dos módulos manipulados na configuração

O penúltimo critério de classificação do processo de configuração diz respeito à complexidade dos módulos manipulados em processos de configuração. Uma configuração em SDRs pode ser classificada de acordo com a dimensão dos módulos manipulados. Uma configuração *incremental* constitui-se na modificação de alguns poucos bits durante o processo de reconfiguração, ao passo que uma configuração *modular* reflete a modificação de um núcleo (i.e. módulo arbitrariamente complexo) inteiro do sistema.

2.1.6 Formato do arquivo de configuração

O último critério de classificação de configuração considera o formato dos dados de configuração. Duas classes são aqui apresentadas: compactada e normal. Uma configuração dita *normal* é aquela onde os dados de configuração são armazenados em sua forma normal, ou seja, uma seqüência de bits de zeros e uns, sem qualquer tipo de compactação. Alguns dispositivos e/ou sistemas reconfiguráveis empregam técnicas de compactação de dados de configuração, ou seja, recebem dados de configuração compactados e possuem algoritmos que descompactam estes dados e realizam a tarefa de configuração a seguir. Neste caso, a configuração é dita *compactada*.

A necessidade de comprimir os dados de configuração cresce à medida que o tamanho de arquivos de configuração cresce além da capacidade de armazenamento em dispositivos de memória de baixo custo. O crescimentos vertiginoso da densidade de dispositivos reconfiguráveis torna difícil armazenar uma única configuração em uma memória "barata" (e.g. flash), pois estas não são encontradas em grandes tamanhos. Não é incomum que

para armazenar uma configuração sejam necessários vários dispositivos. Por outro lado, o uso de configurações compactadas implica o acréscimo de hardware/software para realizar sua compactação/descompactação. Isto gera um compromisso que deve ser avaliado para cada sistema reconfigurável a projetar/implementar.

Na Tabela 2.1 é apresentado um resumo das classificações do processo de configuração propostas neste Capítulo. Os critérios propostos são apresentados à esquerda e suas respectivas classificações aparecem à direita.

Tabela 2.1: Classificações do processo de configuração – *Um processo de configuração pode ser classificado de acordo com seis critérios, como pode ser visto na tabela abaixo.*

Critérios	***Classificações***
1. Largura do caminho de dados de configuração	**a.** Configuração paralela **b.** Configuração serial
2. Comando do processo de configuração	**a.** Configuração autônoma **b.** Configuração dependente
3. Localização da memória de configuração	**a.** Configuração com memória interna **b.** Configuração com memória externa
4. Tamanho do arquivo de configuração	**a.** Configuração total **b.** Configuração parcial
5. Dimensão dos módulos manipulados na configuração	**a.** Configuração incremental **b.** Configuração modular
6. Formato do arquivo de configuração	**a.** Configuração normal **b.** Configuração compactada

2.2 Suporte para reconfiguração dinâmica e parcial

Zhang e Ng apresentam em [ZHA00] o estado da arte em ferramentas para reconfiguração dinâmica e parcial. Uma análise abrangente de suporte de software é apresentada. O trabalho evidencia como principal carência no desenvolvimento de sistemas de natureza reconfigurável: a ausência de ferramentas adequadas. Além disso, Mesquita em [MES02] salienta que a carência de dispositivos comerciais adaptados à reconfiguração parcial é outro fator importante que limita a adoção mais ampla desta tecnologia. Em [COM02], Compton e Hauck apresentam um resumo sobre sistemas parcialmente reconfiguráveis, abordando os principais sistemas de hardware e também ferramentas de software utilizados como suporte ao projeto.

Em virtude do fato de existirem bons trabalhos de revisão do estado da arte em ferramentas para reconfiguração dinâmica e parcial, sobretudo no que tange o suporte de software, no presente documento não é apresentada uma revisão mais ampla de SDRs. Revisa-se nessa Seção algumas ferramentas para geração de configurações parciais, além de métodos para interconexão de núcleos, ambos fatores importantes na realização da fase de síntese física e infra-estrutura de reconfiguração do arcabouço PaDReH, apresentado na Seção 1.4.

2.2.1 Tecnologias habilitadoras

Ainda hoje, são poucas as famílias de dispositivos semicondutores comerciais que habilitam reconfiguração dinâmica e parcial. Como citado em [MES02], os primeiros dispositivos que suportaram reconfiguração parcial são: o Clay fabricado pela National Semiconductor,[NAT98]; o Cal1024 fabricado pela Algotronix, [ALG89]; e o XC6200 fabricado pela Xilinx inc, [XIL97]. Este último ainda é usado no meio acadêmico. Como pode-se notar no Capítulo 3, o XC6200 foi usado como dispositivo alvo para a modelagem de algumas das principais propostas de controladores de configurações. Aparentemente, devido ao suporte precário, sobretudo de software para a síntese física destes dispositivos, eles não foram tão difundidos comercialmente e, não obtiveram sucesso comercial, muito embora tenham sido amplamente utilizados no meio acadêmico. Todas essas famílias iniciais estão hoje descontinuadas.

Atualmente, dois fabricantes comercializam dispositivos que habilitam reconfiguração dinâmica e parcial. São eles a Atmel Corp. e a Xilinx Inc. A Atmel fabrica duas famílias que possibilitam reconfiguração dinâmica e parcial denominadas AT6000 e AT40k. A série AT40K [ATM00] implementa a técnica denominada *Cache Logic*, um termo proposto pelo fabricante para designar a capacidade de reconfiguração dinâmica e parcial. Esta técnica permite que funções sejam substituídas em tempo de execução no FPGA, enquanto o sistema continua a operar [ATM00a], ou seja RTR. Se novas funções se fazem necessárias, as antigas são sobrescritas, como apresentado no diagrama contido na Figura 2.1.

A família AT6000 [ATM00b] foi projetada para acelerar o desempenho de sistemas baseados em processadores, enquanto diminui o custo e o consumo de energia. O número massivo de registradores permite seu uso como co-processadores reconfiguráveis para Processadores de Sinais Digitais (em inglês, *Digital Signal Processors* ou DSPs). Assim como a família AT40K, a família AT6000 também possibilita a implementação de projetos valendo-se do conceito de *Cache Logic*.

Dispositivos da família AT40K possuem entre cinco mil e cinqüenta mil portas lógicas equivalentes e os da família AT6000 possuem entre seis mil e trinta mil portas lógicas equivalentes. Usa-se o termo equivalente porque cada fabricante usa métodos diferentes para medir a densidade de seus dispositivos. Desta forma, uma porta lógica equivalente de dispositivos da Atmel não reflete a mesma densidade que uma porta lógica em dispositivos da Xilinx ou da Altera. Em [WAL02], Waller evidencia a diferença entre os métodos usados para medir o número de portas lógicas por diferentes fabricantes. Por exemplo, estima-se hoje que uma porta lógica para dispositivos APEX Altera equivale a duas portas lógicas para dispositivos Virtex da Xilinx.

A Xilinx Inc., uma das empresas que dominam o mercado de dispositivos configuráveis do tipo FPGA, comercializa pelo menos três famílias de dispositivos que suportam reconfiguração dinâmica e parcial. São elas: Virtex, VirtexII e VirtexII-Pro.

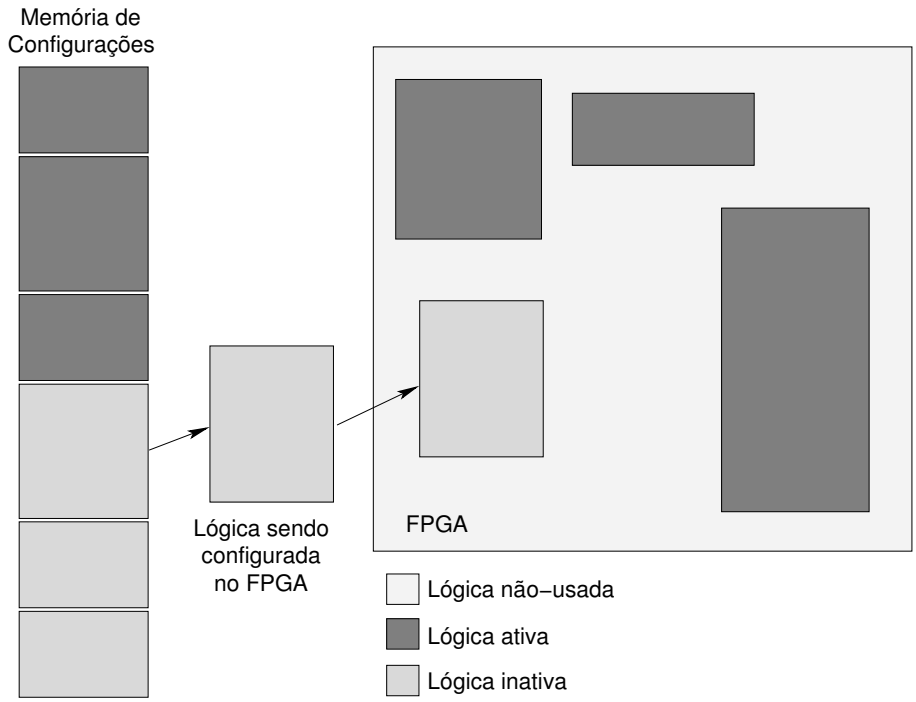

Figura 2.1: Diagrama da *Cache Logic – Cache Logic é a característica de alguns FPGAS da Atmel de armazenar diversos arquivos de configuração internamente e substituir uma configuração pela outra. Esta técnica permite reconfiguração do circuito em tempo de execução, de acordo com uma troca de contexto ou uma eventual modificação do fluxo de entrada de dados. Desta forma, pode-se implementar hardware virtual.*

Os FPGAs da família Virtex são compostos por blocos lógicos configuráveis (CLBs), blocos de entrada e saída (IOBs), blocos de memória RAM (BRAMs), recursos de relógio e roteamento programável, todos configuráveis. Para configurar um dispositivo da família Virtex deve-se preencher uma memória de configuração que determina o comportamento dos elementos componentes do dispositivo.

A memória de configuração pode ser vista como uma matriz bidimensional de bits. Estes bits são agrupados em quadros verticais com um bit de largura, e se estendem verticalmente na forma de colunas que atravessam todo o dispositivo. Um *quadro* é, portanto, a unidade atômica de configuração, ou seja, é a menor porção de memória de configuração que pode ser lida ou escrita. Cada coluna de recursos é dividida em um certo número de quadros.

O maior dispositivo da família Virtex permite implementar um circuito contendo um milhão de portas lógicas equivalentes, de acordo com a Tabela 2.2. Nota-se que a densidade de dispositivos desta família é muito superior à apresentada por dispositivos Atmel. Para maiores informações sobre a família Virtex da Xilinx remete-se o leitor a [XIL01]. Os detalhes de configuração de dispositivos pertencentes a esta família foram explorados e

devidamente documentados por Mesquita em [MES02].

Os FPGAS da família VirtexII [XIL02b] permitem implementar circuitos entre quarenta mil e oito milhões portas lógicas equivalentes, como pode ser visto na Tabela 2.2. Dispositivos dessa família, além dos componentes básicos existentes na família Virtex, possuem colunas de multiplicadores de 18x18 bits. Os detalhes de configuração desta família são precariamente documentados pelo fabricante. Algumas informações a respeito dessa podem ser obtidas em [XIL02c]. Os competidores mais próximos dos FPGAS da família VirtexII são os da família Stratix-Gx [ALT03] da Altera, que possui capacidade equivalente àqueles mas não habilita reconfiguração dinâmica e parcial.

Os FPGAS da família VirtexII-Pro [XIL03] constituem uma extensão da família VirtexII. A novidade desta família é conter processadores IBM PowerPC 405 embarcados. Segundo os fabricantes, pela primeira vez os projetistas podem particionar seus sistemas entre hardware e software durante o ciclo de desenvolvimento de uma maneira mais flexível, ou seja, não apenas no início do projeto. O maior dispositivo desta família possui quatro processadores PowerPC embarcados. Os FPGAs da família VirtexII-Pro são aqueles de maior densidade e maior desempenho na atualidade.

Na Tabela 2.2 apresenta-se um quadro comparativo entre alguns dos dispositivos que habilitam reconfiguração dinâmica e parcial. Nesse comparativo pode-se perceber que os dispositivos das famílias Virtex, VirtexII e VirtexII-Pro comercializados pela Xilinx apresentam uma densidade maior em comparação aos comercializados pela Atmel. As densidades dos dispositivos da família VirtexII-Pro foram obtidas através de estimativas sem considerar os processadores PowerPC embarcados. Considerando a diferença entre número de CLBs dos dispositivos das famílias VirtexII e VirtexII-Pro pode-se estimar qual a diferença entre suas densidades, considerando também o incremento de blocos multiplicadores e blocos de memórias.

A plataforma V2MB1000 da Memec-Insight

A arquitetura alvo para implementação dos sistemas apresentados nesse documento compõe-se de dispositivos da família Virtex-II [XIL02c] fabricados pela empresa Xilinx, devido:

- ao fato destes encontrarem-se disponíveis no ambiente onde o trabalho proposto foi desenvolvido;
- sua densidade habilita prototipar SoCs, ao contrário de dispositivos Atmel;
- apesar de pouco, existe algum suporte à reconfiguração dinâmica e parcial. [XIL00, XIL02d].

Apenas os dispositivos reconfiguráveis não são suficientes para se possa implementar sistemas reconfiguráveis dinâmica e parcialmente. Necessita-se também de interfaces para

Tabela 2.2: Quadro comparativo entre famílias de dispositivos que habilitam RTR – *Nota-se que os dispositivos das famílias Virtex, VirtexII e VirtexII-Pro comercializados pela Xilinx apresentam uma maior densidade em comparação aos comercializados pela Atmel. O dispositivo em destaque, XC2V1000 da família VirtexII, é utilizado como base para o desenvolvimento dos estudos desse trabalho.*

	Dispositivo	***Portas Lógicas***	***Células***	***Linhas x Colunas***
AT40k	AT40K05	5k - 10k	256	16 x 16
	AT40K10	10k - 20k	576	24 x 24
	AT40K20	20k - 30k	1.024	32 x 32
	AT40K40	40k - 50k	2.304	48 x 48
AT6000	AT6002	6k	1.024	32 x 32
	AT6003	9k	1.600	40 x 40
	AT6005	15k	3.136	56 x 56
	AT6010	30k	6.400	80 x 80
Virtex	XCV50	57.906	1.728	16 x 24
	XCV100	108.904	2.700	20 x 30
	⋮			
	XCV800	888.439	21.168	56 x 84
	XCV1000	1.124.022	27.648	64 x 96
VirtexII	XC2V40	40k	576	8 x 8
	XC2V80	80k	1.152	16 x 8
	⋮			
	XC2V1000	**1M**	**11.520**	**40 x 32**
	⋮			
	XC2V6000	6M	76.032	96 x 88
	XC2V8000	8M	104.832	112 x 104
VirtexII-Pro	XC2VP2	80k	3.168	-
	XC2VP4	160k	6.768	-
	⋮			
	XC2VP100	8M	99.216	-
	XC2VP125	10M	125.136	-

controlar externamente estes dispositivos e meios para verificar seu correto funcionamento, ou seja, uma forma de implementar e validar o sistema. Plataformas de prototipação são um meio de conseguir realizar experimentos com tecnologia dinâmica e parcialmente reconfiguráveis.

Especificamente, a plataforma V2MB1000 da Memec-Insight [MEM02], usada nesse trabalho possibilita o desenvolvimento de sistemas baseado na família de FPGAs VirtexII da Xilinx. Além de um dispositivo dessa família, a plataforma contém outros recursos que propiciam ao usuário a oportunidade de desenvolver sistemas de diferentes naturezas conforme ilustrado na Figura 2.2. Estes incluem:

- 32 MBytes de memória SDRAM DDR, com organização 16M x 16;

- memória ISP PROM XC18V04 de 4Mbits utilizada principalmente para configurar o FPGA no modo autônomo paralelo/serial;
- geradores de relógio baseados em cristal de 100MHz e 24MHz;
- porta serial RS232;
- porta JTAG, para configuração do dispositivo;
- porta SelectMAP, para configuração do dispositivos em modos autônomo/dependente paralelo;
- interface LVDS para transmissão de dados em alta velocidade, na ordem de Gbps;
- dispositivos de interface com usuário como: dois *displays* de sete segmentos, oito chaves e quatro botões.

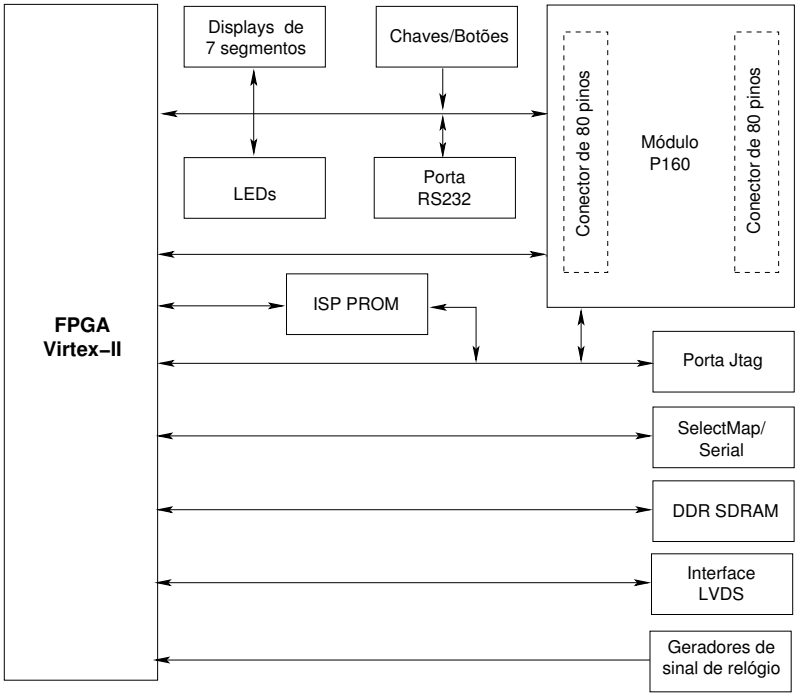

Figura 2.2: Diagrama de blocos da plataforma V2MB1000 – *A plataforma comercializada pela empresa Memec-Insight contém um FPGA da família VirtexII, memórias e interfaces com o usuário.*

Através do módulo de expansão P160 [MEM02a], uma placa de expansão incluída na plataforma, pode-se tanto ter acesso a pinos do FPGA quanto ter acesso a diversas interfaces de comunicação, utilizando-se o módulo de comunicação [MEM02a]. Dentre as possíveis interfaces pode-se citar: USB, Ethernet, RS232 serial, PS/2 para teclado etc. Este módulo ainda contém memórias Flash 2M x 32 e SRAM 256K x 32.

O kit comercializado pela empresa Memec-Insight possui, além da placa principal e módulos de expansão P160, um *firmcore* de um processador denominado MicroBlaze da

Xilinx. Esse processador possui instruções que manipulam dados de 32 bits em dois modos de endereçamento e 32 registradores de propósito geral. O processador MicroBlaze possui uma arquitetura Harvard, ou seja, trabalha com memórias de dados e instruções separadas. Juntamente com o *firmcore* do processador, tem-se acesso a um ambiente de desenvolvimento denominado EDK [XIL03a] com ferramentas de compilação e depuração de programas descritos em linguagem C bem como ferramentas de parametrização do processador e de seus periféricos.

Em um sistema que utiliza o MicroBlaze os núcleos externos a estes podem se conectar ao processador através do CoreConnect da IBM [IBM99], uma arquitetura de barramentos que permite comunicação entre o processador e periféricos. Conforme a especificação do barramento de periférico interno ao circuito (em inglês, *On-chip Peripheral Bus* ou OPB) o núcleo possui barramentos separados de 32 bits de instrução e dados. O processador pode acessar diretamente os blocos de memória RAM do dispositivo, utilizando o barramento da memória local (em inglês, *Local Memory Bus* ou LMB). MicroBlaze suporta multiplicações em hardware quando é implementado sobre plataformas da família VirtexII e VirtexII-Pro, os quais possuem blocos específicos para realizar esse tipo de operação como apresentado anteriormente. Para maiores detalhes sobre o processador MicroBlaze, remete-se o leitor a [XIL02, XIL02a, XIL03].

2.2.2 Manipulação de arquivos de configuração

A geração de arquivos de configuração parciais apresenta-se como uma tarefa crucial para o desenvolvimento de SDRs. Os fabricantes fornecem junto com seus dispositivos ambientes de CAD para desenvolvimento de sistemas que contêm, muitas vezes, centenas de ferramentas. Dentre estas, algumas se dedicam à geração de arquivos de configuração parciais. A Xilinx, por exemplo, fornece o ambiente ISE aos seus usuários. Para criar um arquivo de configuração parcial o projetista pode usar a ferramenta BitGen, mas antes deve empregar no mínimo outras duas ferramentas. A primeira delas é o Floorplanner para determinar a geometria da área do FPGA que conterá o módulo reconfigurável. A segunda ferramenta é o FPGAEditor, usado para gerar arquivos parciais com modificações. BitGen é um software usado apenas para gerar o arquivo binário parcial contendo a diferença entre um arquivo de configuração total original e um arquivo que contém informações relativas às modificações realizadas no FPGAEditor.

O fluxo de geração de arquivos parciais com as ferramentas fornecidas pelos fabricantes deve ser realizado, ainda hoje de forma manual. Por isso, trata-se de um processo demorado e complexo, passível de erros, para projetos que necessitam de diversas configurações parciais. Necessita-se, portanto, de ferramentas mais automatizadas para o projeto de sistemas reconfiguráveis.

Nessa Subseção apresentam-se algumas ferramentas que tem por objetivo auxiliar projetistas na tarefa de gerar arquivos de configuração parciais.

PARBIT

Na Universidade de Washington foi desenvolvido um ambiente que visa facilitar a configuração parcial de FPGAs Xilinx, denominado PARBIT (do inglês *PARtial BItfile Transformer*) [HOR02a, HOR01]. A partir da execução de um conjunto de programas, o usuário pode gerar arquivos parciais para diferentes partes de um FPGA. São passados à ferramenta PARBIT, como argumentos, arquivos com opções do usuário escolhendo um dispositivo alvo, a localização para inserção do módulo reconfigurável e a descrição do mesmo. PARBIT pode gerar arquivos apenas para dispositivos da família VirtexE da Xilinx, composta por FPGAs semelhantes aos da família Virtex com recursos de memórias mais extensos.

A mesma equipe de pesquisadores propôs o conceito de *Dynamic Hardware Plugins* (DHP) [HOR02, TAY01, TAY02]. DHPs permitem que múltiplas aplicações de hardware, ou *plugins*, sejam dinamicamente carregada em um único dispositivo e executem em paralelo. Em outras palavras, essa técnica pode ser utilizada para implementar SDRs.

Um problema enfrentado na geração de arquivos parciais é controlar o processo de roteamento. Por isso é importante notar, em [HOR02], a existência de uma colaboração direta com a empresa Xilinx, que adaptou uma ferramenta de roteamento para que esta aceite a imposição de restrições que viabilizam a reconfiguração parcial de maneira mais automatizada.

JBits

JBits [GUC99] é uma API Java que fornece acesso direto a arquivos de configuração de dispositivos Virtex da Xilinx. Esse conjunto de classes *Java* é fornecido pela Xilinx para que desenvolvedores possam criar seus próprios programas para manipular arquivos de configuração de dispositivos por ela fabricados. As modificações em arquivos de configuração incluem manipulação de funções lógicas do dispositivo e manipulação do roteamento. O fato de o arquivo poder ser modificado diretamente possibilita o desenvolvimento de SDRs.

JPG

Em [RAG02], Raghavan e Sutton apresentam um outro ambiente para geração de arquivos de configuração parciais, denominado JPG. Ele é implementado em *Java* e emprega a API JBits. O ambiente JPG usa arquivos no formado XDL (em inglês, *Xilinx Design Language*) gerados por aplicativos fornecidos pela Xilinx e UCF (em inglês, *User Constraints File*) para restringir a área a ser usada no roteamento de configurações parciais. Este mesmo ambiente estabelece uma ligação direta entre o sistema de desenvolvimento ISE e JBits, facilitando a tarefa de geração de arquivos parciais, dependendo, portanto, da API fornecida pela Xilinx.

JRTR

JRTR (em inglês *Java Run-Time Reconfiguration*) [MCM00] é uma extensão da API JBits que fornece suporte à reconfiguração parcial. JRTR combina técnicas de hardware e programas que permitem modificações pequenas em arquivos de configuração de dispositivos da família Virtex de forma direta, rápida e sem interrupção de operação. JBits é utilizada para leitura e escrita de arquivos de configuração. JRTR possui um *parser* para análise do arquivo de configuração. Segundo [MCM00], utilizando JRTR, o usuário pode gerar configurações parciais em qualquer instante, e então configurá-los no dispositivo durante a operação do sistema.

Ferramentas desenvolvidas no GAPH

Em [MöL03], Möller apresenta uma série de ferramentas para manipular arquivos de configurações. Essas ferramentas foram desenvolvidas no mesmo grupo de pesquisa, GAPH (Grupo de Apoio ao Projeto de Hardware), onde o presente trabalho se desenvolve. Algumas das ferramentas desenvolvidas por Möller baseiam-se na API JBits. A seguir apresentam-se, de maneira sucinta, três ferramentas desenvolvidas no GAPH:

- *BitProgrammer* permite acessar manipular arquivos de configuração local ou remotamente. Com essa ferramenta é possível visualizar o conteúdo dos arquivos de configuração bem como modificá-los. Os arquivos também podem ser configurados no dispositivo remotamente.

- *BitAnalyzer* é utilizado para a análise de arquivos de configuração. é possível selecionar partes de um arquivo de configuração total e gerar um arquivo parcial.

- *CoreUnifier* é utilizado para manipular núcleos de propriedade intelectual. Essa ferramenta permite a leitura de informações de configurações diretamente do arquivo de configuração. Um arquivo de configuração pode ser criado a partir da união de áreas selecionadas de dois arquivos de configuração.

Essas, entre outras ferramentas desenvolvidas no grupo, habilitam a manipulação de arquivos de configuração totais e parciais. O conjunto desenvolvido por Möller manipula apenas arquivos de configuração gerados para dispositivos da família Virtex da Xilinx.

Alternativas para geração de arquivos de configuração parciais

Dyer et al. [DYE02] apresentam três alternativas para a geração de arquivos de configuração parciais. A primeira delas emprega fluxos realizados usando apenas ferramentas fornecidas por vendedores de dispositivos reconfiguráveis. Estas são restritas, permitem manipulações incrementais, ou seja, de pequeno porte. Como mencionado anteriormente, essas ferramentas são, ainda hoje, manuais e enfrentam problemas relativos à limitação do roteamento. Em [BRI03], Brião apresenta um trabalho que emprega apenas as ferramentas fornecidas pelo fabricante no desenvolvimento de SDRs.

A segunda alternativa evidenciada por Dyer et al. é usar a API JBits para gerar arquivos parciais. Segundo o mesmo autor, esta alternativa permite um alto grau de abstração, mas não fornece opções de síntese avançadas que permitam realizar otimizações relativas à potência dissipada e temporização, por exemplo.

A terceira alternativa apresentada em [DYE02], segundo os autores a melhor delas, reflete a idéia de combinar as duas anteriores, aproveitando as melhores características de cada. Esse fluxo misto propõe a utilização das ferramentas fornecidas pelo fabricante para realizar as tarefas de síntese e o JBits para gerar os arquivos parciais.

2.2.3 Métodos de RTR dirigidos a núcleos IP

Com a ascensão do desenvolvimento de SoCs, espera-se que cresça a utilização de núcleos IP (em inglês *Intellectual Property*) e também o reuso de hardware. Os desenvolvedores devem poder agrupar diversos núcleos em um único sistema, de maneira mais automatizada possível para atender a demanda do mercado. Para isso, faz-se necessário a disponibilização de núcleos de propriedade intelectual e o emprego de estratégias de conexão de núcleos.

A pouco tempo atrás, desenvolvia-se SoCs de maneira que os núcleos deviam se adaptar a um meio de comunicação padrão anteriormente projetado. Modernamente, além do meio de comunicação, também as interfaces do núcleo podem ser padronizadas. Ost, em [OST03], apresenta pesquisas relacionadas a padronização da interface de comunicação entre núcleos. A seguir, apresentam-se algumas alternativas propostas tendo em vista meios de conexão padronizados para núcleos de propriedade intelectual.

Barramento de Palma

Palma [PAL02] apresenta uma proposta de métodos para desenvolvimento e comunicação de núcleos de hardware no contexto de reconfiguração dinâmica e parcial. O fluxo de desenvolvimento proposto sofre com o problema de falta de suporte para restringir o roteamento de arquivos de configuração parciais, anteriormente citado. A maior contribuição do trabalho de Palma é a proposta de um barramento para interconexão de núcleos de hardware dinâmica e parcialmente reconfiguráveis. Define-se, nesse trabalho, um hardware fixo chamado de *controlador* que coordena a comunicação com o mundo externo e entre os núcleos, virtualizando os pinos de entrada e saída. A comunicação entre o controlador e os núcleos é realizada através de pinos implementados usando duas camadas de *buffers tristate*, uma pertencendo ao controlador e outra ao núcleo conectado ao barramento. O controlador é na verdade um árbitro do barramento. O barramento localiza-se horizontalmente na parte inferior do dispositivo. Na Figura 2.3 pode-se ver a estrutura do barramento proposto por Palma.

Como são usados *buffers tristate*, o número de componentes desta natureza pode vir a limitar o número de linhas (largura) do barramento. A definição do barramento exigiu

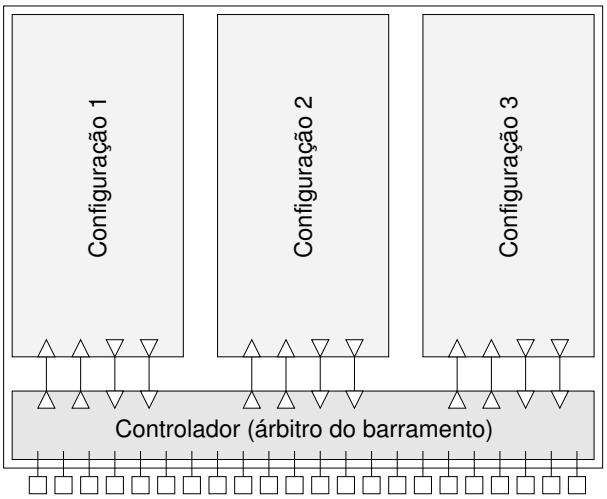

Figura 2.3: Proposta de comunicação entre núcleos de [PAL02] – *O barramento proposto por Palma é composto por um árbitro que controla o acesso ao barramento. Existe uma camada de buffers tristates no módulo reconfigurável e outra equivalente no controlador. A ligação entre as duas camadas buffers tristates é realizada através de barramentos de roteamento comuns. Na parte inferior dessa Figura pode-se notar que os pinos de entrada/saída são ligados ao controlador e não às configurações.*

a definição de um protocolo de comunicação. Com isso, para conectar um módulo no barramento deve ser realizada uma tarefa de empacotamento do módulo. Como citado em [PAL01], faz-se necessária a presença de um programa para empacotar núcleos (empacotador ou *wrapper*) de maneira que todos possuam a mesma interface de comunicação.

É importante salientar que muitas das características estruturais do barramento proposto relacionam-se ao fato deste ser desenvolvido tendo como dispositivo alvo FPGAs da família Virtex da Xilinx. A restrição na largura, por causa dos buffers tristate também se relaciona ao dispositivo utilizado.

Fluxo de projeto modular de Lim

Uma outra proposta para conexão de núcleos sugerida pela Xilinx é encontrada em [LIM02]. Nessa referência são evidenciados dois fluxos para realização de configuração parcial: o primeiro deles representa reconfigurações onde são reconfigurados apenas alguns bits, reconfiguração parcial do tipo *incremental* de acordo com a Tabela 2.1. O segundo fluxo representa a reconfiguração de toda uma parte da lógica, reconfiguração parcial do tipo *modular*, segundo a mesma Tabela. Neste último caso, ressalta-se a presença de uma estrutura para comunicação de núcleos (denominados em [LIM02] módulos) quando existe a necessidade destes se comunicarem uns com os outros. Esta estrutura chama-se *Bus Ma-*

cro, um módulo composto por quatro fios para comunicação, e outros quatro para controle de fluxo como o apresentado na Figura 2.4. Quando são necessários mais que quatro fios para comunicação entre dois núcleos, são usadas quantas Bus Macros forem necessárias. Na Figura 2.4 apresenta-se a estrutura de conexão entre dois módulos utilizando Bus Macro.

A Bus Macro, ao contrário do barramento apresentado por Palma, posiciona-se verticalmente entre módulos reconfiguráveis e/ou fixos, ao longo do dispositivo. A Bus Macro não necessita de um árbitro centralizado, pois interconecta apenas núcleos ponto a ponto. Quando dois núcleos devem se comunicar, isto ocorre através da Bus Macro que os interliga. A Bus Macro, assim como o barramento proposto por Palma, usa *buffers tristate*.

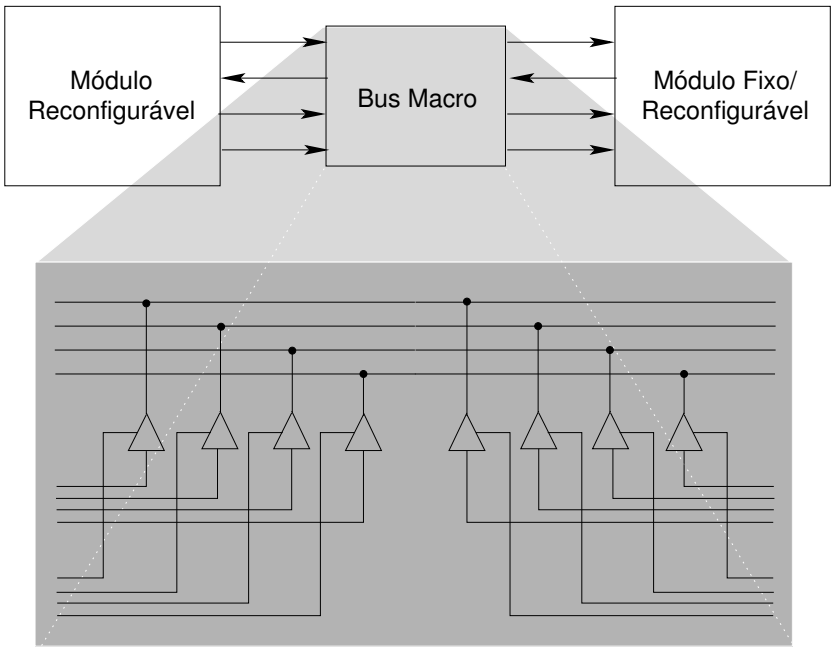

Figura 2.4: Proposta de comunicação entre núcleos de [LIM02] – *As bus-macros devem ser localizadas no limite entres o módulo fixo e o reconfigurável ou entre dois módulos reconfiguráveis de maneira a possibilitar a comunicação entre os mesmos.*

É importante observar que em [LIM02] recomenda-se o desenvolvimento de sistemas usando Bus Macro com apenas dois núcleos: um estático e outro reconfigurável. Não é mencionado que não se pode desenvolver sistemas com número de módulos maior que dois, mas tais recomendações indicam que o modelo possui restrições de uso, ainda que transitórias. Além disso, assim como o fluxo de projeto de sistemas reconfiguráveis de [PAL01], o fluxo proposto em [LIM02] também apresenta várias tarefas manuais que inviabilizam o desenvolvimento de sistemas mais complexos, ou seja, existe ainda carência de automatização do processo de geração de arquivos parciais de configuração.

As informações apresentadas em [LIM02] refletem uma proposta de como utilizar técnicas de projeto modular para implementar sistemas RTR. Em sua essência, a técnica de projeto modular foi concebida para suportar o desenvolvimento de projetos com grande complexidade, que necessitam de diversos grupos de desenvolvedores, muitas vezes em locais geograficamente distribuídos, cada um desenvolvendo uma parte do sistema.

No próximo Capítulo apresenta-se considerações relacionadas ao controle dos processos de reconfigurações dinâmicas e parciais em sistemas dinamicamente reconfiguráveis. Trata-se algumas propostas de estrutura de controladores bem como implementações de módulos dessa natureza.

Capítulo 3

Controladores de configurações

Os desenvolvimentos na área de sistemas reconfiguráveis tem conduzido a hardware que se assemelha a software do ponto de vista de flexibilidade. Em virtude disso hardware reconfigurável necessita de subsistemas que executem operações semelhantes às de um sistema operacional. Como citado anteriormente nesse volume, um sistema operacional gerencia a operação de processos em uma UCP, controlando a carga de programas a executar em memória. Para isto, deve interpretar informações provenientes de um escalonador de processos. Em SDRs, deve existir um subsistema que realize tarefas semelhantes a estas.

Como visto na Seção 1.2.1, pode-se classificar um processador convencional como um hardware reconfigurável onde, cada uma das instruções a ser executada reflete uma configuração diferente. Desta forma, ao executar uma instrução de soma um processador é configurado para executá-la, o que determina qual deve ser seu comportamento e assim ocorre para cada uma das instruções de sua arquitetura.

Em [BRE01], são discutidos métodos para gerenciamento de configurações internos a um CI, ou seja, que não necessitam de um hospedeiro externo. Tais métodos são denominados gerenciamento intra-chip (em inglês, *on-chip management*). Em tais sistemas, o controle de configuração realiza uma operação denominada *autoconfiguração*, termo este definido por Blodget et al. [BLO03]. Segundo este mesmo autor, uma autoconfiguração estende o conceito de reconfigurabilidade dinâmica, pois assume que circuitos específicos no FPGA são usados para controlar a reconfiguração de outra parte do FPGA.

Segundo [BRE01], Controladores de Configurações tradicionais necessitam de um hospedeiro externo ao dispositivo reconfigurável para seu controle. Mas, em um sistema fechado, onde todas as configurações e as possíveis seqüências de execução são conhecidas, um controlador interno ao dispositivo pode ser desenvolvido para gerenciar/controlar o processo de reconfiguração.

Nesse Capítulo, serão apresentados três propostas de modelos de controladores de con-

figuração (Seção 3.2), seguido da discussão de duas implementações (Seção 3.1). A seguir, a Seção 3.3 apresenta um resumo de características gerais de controladores de configuração e a comparação dos modelos e implementações estudados sob o ponto de vista destas características. Outras considerações sobre controladores de configurações, incluindo escalonamento de configurações e o particionamento de dispositivos reconfiguráveis finalizam esse Capítulo.

3.1 Modelos de controladores de configurações

Nessa Seção apresenta-se um breve resumo de trabalhos que propõem modelos de controladores de configuração. Dentre eles os apresentados por Shirazi [SHI98], Burns [BUR97] e Lysaght [MCG99]. Estes trabalhos demonstram a importância de subsistemas para controle de configurações.

3.1.1 Modelo de Shirazi et al.

Shirazi e colaboradores [SHI98] propõem um modelo genérico de controlador de configurações. Esse modelo, apresentado na Figura 3.1, compõe-se de três partes principais (ou estágios): um *carregador de configurações*, um *monitor* e um *armazenamento de configurações*.

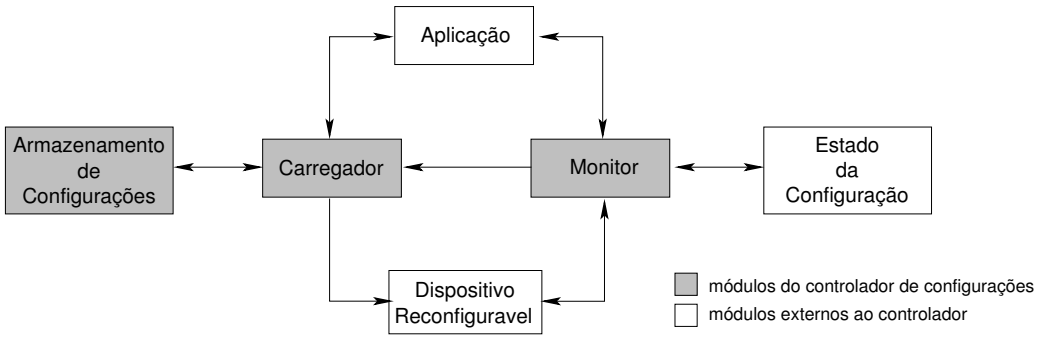

Figura 3.1: Estrutura de um controlador de configurações proposto em [SHI98] – *Este modelo genérico é composto pelos módulos carregador de configurações, monitor e por um armazenamento de configurações.*

Neste modelo, o monitor mantém informações sobre o estado da configuração, tais como tipo e posicionamento no dispositivo reconfigurável. Ele também mantém informações sobre as possíveis transições para o próximo estado de configuração a partir do estado atual. O monitor ainda verifica condições da aplicação que ativam reconfigurações. Se uma condição é satisfeita e a configuração necessária não se encontra disponível no dispositivo, então o monitor notifica o carregador para configurá-la.

Existem três possibilidades de operação do monitor, dependendo das informações disponíveis na seqüência de reconfiguração:

a) O tempo de execução de uma configuração é conhecido em tempo de compilação e a próxima configuração também é conhecida;
b) Apenas a próxima configuração é conhecida;
c) O tempo de execução da configuração e a próxima configuração não são conhecidos.

No primeiro caso, o monitor é composto apenas por um temporizador, que indica quando uma nova configuração deve ser carregada. No segundo caso, são necessárias condições recebidas do dispositivo (e.g. um FPGA) ou da aplicação para que a próxima configuração seja carregada. O último caso ainda necessita que as configurações sejam produzidas em tempo de compilação ou de execução.

O carregador, como seu próprio nome indica, é responsável por transportar configurações para dentro do dispositivo. Quando recebe uma requisição do monitor, o carregador obtém a localização da configuração requisitada no armazenamento de configurações e então inicia o processo de configuração.

O armazenamento de configurações possui três componentes: um diretório de configurações, um repositório de dados de configuração e um agente de transformação. O diretório é uma tabela contendo apontadores para o início de cada configuração no repositório de dados de configuração, que armazena as configurações em si. O agente de transformação é necessário para minimizar o armazenamento de configurações já que ele encarrega-se de realizar a relocação dos arquivos para diversas posições, tornando desnecessário armazenar vários arquivos com a mesma lógica para cada uma das possíveis posições de uso.

A estrutura da Figura 3.1 apresenta o modelo genérico proposto, que contudo pode ser adaptado/personalizado para uma aplicação específica. Esta constitui uma proposta um tanto simplificada da estrutura completa de um controlador de configurações, carecendo de maiores detalhamentos.

3.1.2 Modelo de Burns et al.

O Grupo de Arquitetura Reconfigurável RAGE (*Reconfigurable Architecture Group*), da Universidade de Glasgow desenvolve pesquisas que visam analisar as aplicações de sistemas reconfiguráveis. Em [BUR97], Burns et al. descrevem a estrutura de um SDR denominado RAGE. O fluxo de dados deste sistema é esboçado na Figura 3.2.

O controlador de hardware virtual provê a principal interface entre a aplicação e o sistema em execução. Chamadas para realizar uma reconfiguração ou liberar determinada área ocupada por uma configuração são enviadas ao controlador de hardware virtual, que mantém um mapa da utilização do dispositivo.

O controlador de transformações é responsável por adaptar uma configuração para uma dada localização no dispositivo, se a posição original para qual esta foi projetada estiver

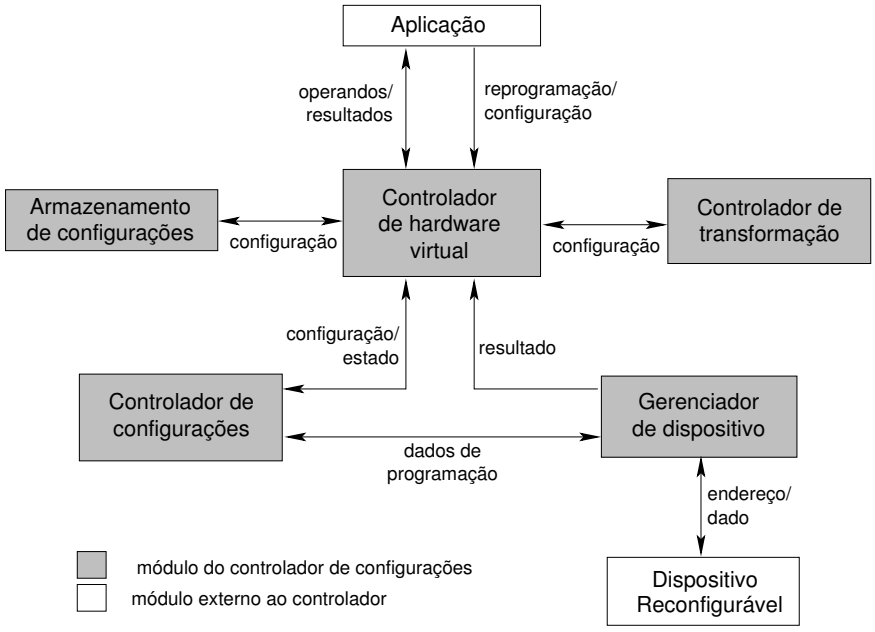

Figura 3.2: Fluxo de dados do sistema RTR RAGE [BUR97] – *A estrutura de sistema RTR segundo [BUR97] é composta por: controlador de hardware virtual, controlador de transformações, controlador de configurações e o gerenciador de dispositivos.*

ocupada por outra configuração, realizando assim a tarefa de relocação de configurações.

O controlador de configurações provê uma interface para o controlador de hardware virtual independente do dispositivo. Ele realiza alguns dos seguintes serviços:

- Realiza reconfigurações através de um gerenciador de dispositivo específico;
- Carrega dados passados da aplicação, através do controlador de hardware virtual, para registradores das configurações residentes no FPGA;
- Passa informações de estado fornecidas pelo gerenciador do dispositivo para o controlador de hardware virtual.

O gerenciador de dispositivo fornece um conjunto de instruções que possibilitam ao controlador de configurações programar/configurar o dispositivo e comunicar-se com uma placa contendo um ou mais dispositivos reconfiguráveis.

As propostas de [SHI98] e [BUR97] para controladores de configurações, embora genéricas, foram desenvolvidas tendo em mente um caso especial de estrutura de sistema reconfigurável, aquele onde uma aplicação (ver Figuras 3.1 e 3.2) corresponde a um software executando em um processador hospedeiro, o controlador de configurações sendo parte do software executando no mesmo hospedeiro e o dispositivo reconfigurável sendo um co-processador usado pela aplicação, normalmente residindo em um periférico do hospedei-

ro. Entretanto, nenhuma característica dos modelos citados implica que os controladores propostos não possam ser implementados em hardware.

3.1.3 Modelo de Lysaght et al.

McGregor e Lysaght classificam controladores de configurações em duas categorias [MCG99]: implementados em hardware ou em software. Os autores salientam ainda que a escolha da melhor forma depende da natureza da aplicação reconfigurável. Eles sustentam que um controlador pode também ser uma mescla de software e hardware, ou seja, ter algumas das suas funcionalidades operando em software enquanto outras estão operando em hardware, sendo que estas de alguma forma trocam informações.

Segundo Robinson e Lysaght [ROB99], as sobrecargas de área em hardware ou de temporização introduzidas por um controlador de configurações podem inviabilizar o uso de técnicas de RTR. Estas sobrecargas podem estar associadas aos recursos necessários, ao atraso na execução e também relacionadas ao projeto e testes de um novo controlador para cada novo sistema.

A principal contribuição de [ROB99] é a proposta de um modelo genérico de controlador de configurações, detalhado na Figura 3.3, que pode ser adaptado de acordo com as necessidades do usuário. Neste modelo, uma configuração, denominada pelos autores *tarefa*, pode estar no *conjunto de tarefas ativas* ou no *conjunto de tarefas inativas*. Tarefas *estáticas* estão sempre no primeiro conjunto enquanto que tarefas *dinâmicas* podem entrar e sair do mesmo. O tempo necessário para transferir uma tarefa dinâmica do conjunto de tarefas inativas para o conjunto de tarefas ativas é chamado *latência de reconfiguração* [LYS97]. Essas transferências só ocorrem quando uma condição pré-definida, dita *condição de reconfiguração*, for satisfeita. Nesse caso, os dados de configuração são carregados no dispositivo reconfigurável através da *porta de configuração*.

O modelo de controlador de configurações proposto em [ROB99] baseia-se em blocos que executam operações ditintas. As transferências entre os blocos são controladas por protocolos de comunicação. Os blocos desnecessários em determinados projetos podem ser retirados sem afetar o modelo. Segundo os autores, uma unidade de controle de configurações para SDRs deve realizar três operações principais:

- Identificar condições de reconfiguração;
- Recuperar dados de configuração;
- Configurar o dispositivo usando os dados de configuração.

Além dessas, algumas funcionalidades podem ser desejáveis:

- Disponibilidade de um mecanismo de enfileiramento de condições de reconfiguração;
- Controle de níveis de prioridade para configurações;

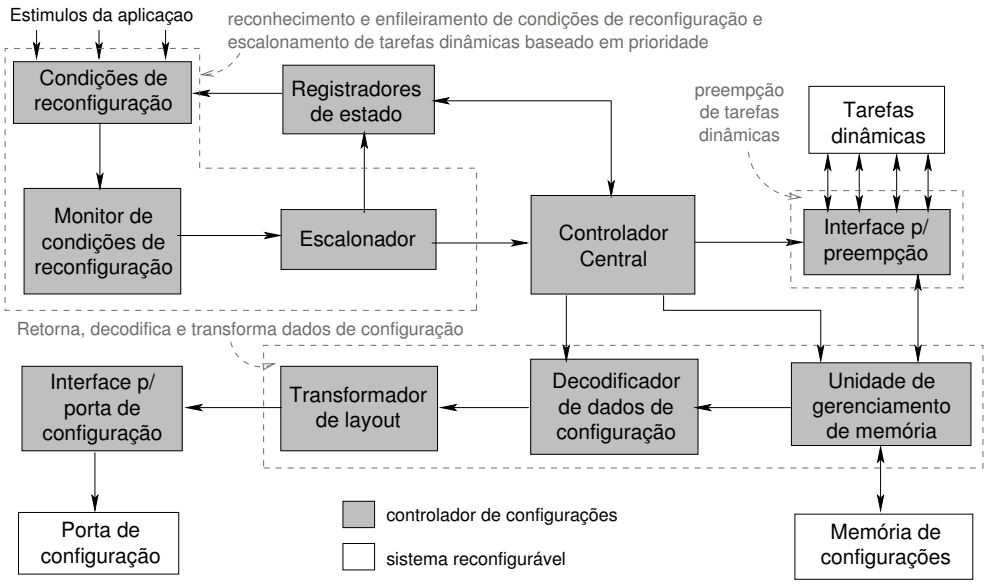

Figura 3.3: Estrutura do controlador de configurações genérico proposto em [ROB99] – *O modelo mais sofisticado de controle de configurações prevê o uso de preempção além da possibilidade de manipular dados compactados ou encriptados. Ele ainda possui um escalonador, que nos outros modelos não faz parte do sistema e apenas fornece entradas para o controlador de configurações.*

- Decodificação de dados de configuração em tempo de execução;
- Transformação em tempo de execução do layout de tarefas dinâmicas, ou seja, relocação.

O controlador central comanda a ativação e a desativação de tarefas dinâmicas. Este processo inicia quando um identificador de tarefa e um sinalizador são gerados na saída do escalonador. O sinalizador indica se a tarefa será ativada ou desativada. O controlador central ativa a nova tarefa dinâmica, preemptando, se necessário, alguma outra tarefa dinâmica. Na desativação, o controlador central restaura alguma outra tarefa dinâmica anteriormente preemptada. O controlador central assegura a consistência dos registradores de estado através desse processo.

Cada tarefa dinâmica tem duas condições de reconfiguração, correspondentes à ativação e desativação. No modelo proposto, pode ser especificado após quanto tempo uma condição de reconfiguração deve ocorrer, entre outros fatores.

O monitor de condições de reconfiguração é responsável por detectar e armazenar a satisfação de condições de reconfiguração para que estas sejam processadas pelo controlador central.

O escalonador serializa requisições paralelas de reconfiguração. Ele recebe dados de filas de prioridade no monitor de condições de reconfiguração e cria uma única fila de saída,

contendo um identificador de tarefa dinâmica e um sinal de controle para cada condição de reconfiguração. As tarefas dinâmicas podem ser selecionadas através de dois esquemas: uma fila circular ou diversas filas com prioridades diferentes acessadas seqüencialmente.

Os registradores de estado são responsáveis por armazenar o estado de cada uma das tarefas dinâmicas. Condições de reconfiguração usam o estado de uma tarefa dinâmica para ordenar o escalonamento de ativação e desativação, ou seja, definir quando tais operações irão ocorrer. Cada tarefa tem a ela associado um registrador de estado que informa se ela encontra-se configurada no dispositivo ou não. O conjunto mínimo de estados suportado por esse modelo é composto por:

- *presente*;
- *preemptada*;
- *reconfigurando*;
- *escalonada para ativação*;
- *escalonada para desativação*.

A interface para preempção gerencia a remoção de tarefas dinâmicas de menor prioridade para dar lugar às de maior prioridade. Para que a tarefa preemptada continue executando a partir de onde parou, depois da tarefa de maior prioridade executar, seu estado deve ser salvo antes da preempção e restaurado no seu retorno ao dispositivo. A própria tarefa executa rotinas internas para salvar seu estado (contexto) antes de "sair" do dispositivo, e sinaliza ao controlador central quando terminou esse processo. Este fator complica o desenvolvimento de tarefas usando esquemas de preempção, uma vez que fica a cargo da tarefa gerenciar seu contexto.

O decodificador de dados de configuração inclui circuitos de configuração definidos pelo usuário para converter os dados de configuração codificados em dados utilizáveis. A codificação aqui relaciona-se à compactação de dados e/ou encriptação.

O transformador de layout tem por objetivo maximizar a utilização do dispositivo configurável. Isso pode ser alcançado atrasando a decisão do posicionamento final da tarefa no dispositivo, através de técnicas de relocação de bitstreams, e/ou mediante uso de técnicas de defragmentação de áreas reconfiguráveis.

3.2 Implementações de controladores de configurações

Após a apresentação dos modelos de controladores propostos, descreve-se abaixo algumas implementações de sistemas dessa natureza. Nenhuma destas implementações segue exatamente qualquer dos modelos propostos apresentados acima.

3.2.1 Implementação de Curd

Curd [CUR03] descreve a implementação de um controlador de configurações para dispositivos VirteII-Pro. Este controlador é específico para configurar RocketIO, transceptores de banda larga incorporados a FPGAs VirtexII-Pro. Um esboço da estrutura do controlador desenvolvido pode ser visto na Figura 3.4.

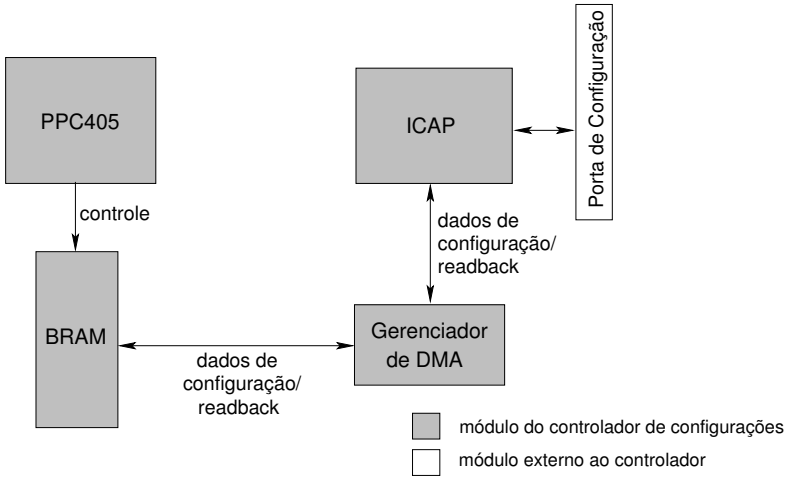

Figura 3.4: Estrutura do controlador de configurações implementado em [CUR03] – *O PPC405 controla o processo de configuração, lendo dados para configuração da memória BRAM e enviando-os para ICAP, uma interface interna de configuração do dispositivo. A escrita/leitura de dados no/do ICAP é controlada pelo gerenciador de DMA.*

O controlador é responsável por comandar o processo de reconfiguração parcial, modificando os atributos do transceptor de banda larga. ICAP (em inglês *Internal Configuration Access Port*, Porta Interna de Acesso à Configuração) é um módulo que fornece acesso interno aos pinos de configuração no modo paralelo de configuração de dispositivos VirtexII e VirtexII-Pro da Xilinx. Através dele, é possível enviar dados de configuração para o FPGA internamente ao próprio dispositivo. Esta operação não podia ser realizada em dispositivos da família Virtex, por exemplo, onde necessitava-se de um módulo externo para acessar a interface de configuração através de pinos específicos do FPGA.

O controlador é composto por quatro módulos. O primeiro deles, denominado ICAP recebe como entrada: um sinal de habilitação (ce); um sinal de seleção da operação a ser realizada ($write$); um sinal de relógio; e um byte de dados a ser enviado para o FPGA (i). Gera sinais de ocupado ($busy$) e um byte de dados (o) lidos a partir do FPGA no caso de *readback* (i.e. ler os bits de configuração do dispositivo, operação inversa à configuração).

O segundo módulo que compõe o controlador de Curd [CUR03] denomina-se BRAM.

Ele é composto por 3 BRAMs de 18Kb: duas para armazenar o código a ser executado pelo processador PPC405 e uma para armazenar os dados de configuração.

O terceiro módulo é o processador PPC405. Principal módulo do sistema, responsável por controlar todo o processo de configuração. Este módulo é implementado em software. Seu código fonte, armazenado nas duas primeiras BRAMs, descreve rotinas para ler dados de configuração da outra BRAM e enviá-los ao gerenciador de DMA.

O gerenciador de DMA (em inglês, *Direct Memory Access*, Acesso Direto à Memória) é o último componente do controlador. Este módulo é responsável por gerenciar a transmissão de dados de configuração entre a BRAM utilizada como repositório de configurações e a interface de configuração do dispositivo, no caso ICAP.

Embora os módulos do sistema sejam estruturas implementadas em hardware, a lógica de controle de configurações é representada por um código executado pelo PPC405. No modelo de Curd [CUR03], o hardware é utilizado para armazenamento temporário de dados e gerenciamentos da transmissão dos mesmos.

Este controlador é idealizado apenas para controlar a configuração de transceptores de banda larga (i.e. RocketIO da VirtexII-Pro). Em face disso, ele não possui componentes para realizar tarefas tais como relocação de configurações e escalonamento entre outras. Limita-se a modificar informações contidas em uma palavra de controle que determina o comportamento dos transceptores RocketIO, recursos fixos no dispositivo. As configurações realizadas por esse controlados são incrementais segundo o critério Dimensão dos módulos manipulados na configuração apresentado na Subseção 2.1.

3.2.2 Implementação de Blodget et al.

Em [BLO03], Blodget et al. descrevem uma proposta semelhante a apresentada na Seção anterior, já que a lógica de controle é implementada em software. A estrutura deste controlador de configurações é ilustrada na Figura 3.5.

O funcionamento desse controlador é comandado por um programa que executa no processador MicroBlaze e requisita um quadro (unidade atômica de configuração de dispositivos da família Virtex e VirtexII da Xilinx) específico. A seguir, a lógica de controle utiliza a interface ICAP para realizar uma operação de *readback* e carregar os dados de configuração em uma BRAM. Quando a operação de *readback* termina, o programa modifica a configuração armazenada na BRAM. Finalmente, o ICAP é utilizado para configurar o quadro com o dado modificado.

O sistema foi desenvolvido para dispositivos da família VirtexII da Xilinx, onde núcleos MicroBlaze podem ser configurados, como mencionado na Subseção 2.2.1. Este modelo configura apenas quadros unitários e, portanto, não necessita de memória externa para

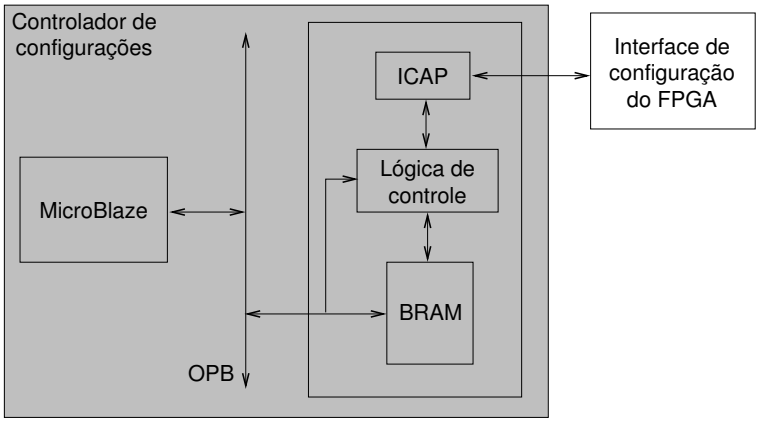

Figura 3.5: Estrutura do controlador de configurações discutido em [BLO03] – *O controlador executando no processador MicroBlaze requisita um quadro de configuração. A partir daí, a lógica de controle realiza uma operação de readback através da ICAP e carrega os dados lidos em uma BRAM. Após o readback, o programa modifica estes dados e os envia à ICAP para reconfigurar dispositivo com a lógica modificada.*

armazenar dados de configuração.

Ainda na Figura 3.5, pode-se notar a existência de um barramento *on-chip peripheral bus* (OPB) do CoreConect. *CoreConnect* [IBM99] é uma interface de comunicação baseada na arquitetura de barramento desenvolvida pela IBM.

O sistema de software deste controlador foi implementado em camadas para facilitar sua modificação. Apresenta-se um esboço da estrutura de software em camadas na Figura 3.6.

A camada 3 é responsável pela interface com a Aplicação embarcada. A camada 2 comporta APIs semelhantes a JBits que possibilitam configurar ou realizar *readback* do dispositivo de forma parcial. A camada 1 possui controladores de dispositivo embarcados no MicroBlaze e um emulador de controlador de ICAP executando em um computador hospedeiro. A camada 0 (zero) constitui-se do controlador de ICAP. As camadas 2 e 3 são independentes do hardware enquanto que as demais, 0 e 1, são dependentes.

3.3 Considerações em controladores de configurações

Os modelos e implementações estudadas até o presente momento diferem em sua complexidade. Contudo, todos possuem blocos (ou conjuntos de blocos) que desempenham um mesmo conjunto de tarefas. Em todos os casos existe:

- um meio de armazenamento de configurações, com exceção da implementação de Blodget [BLO03];

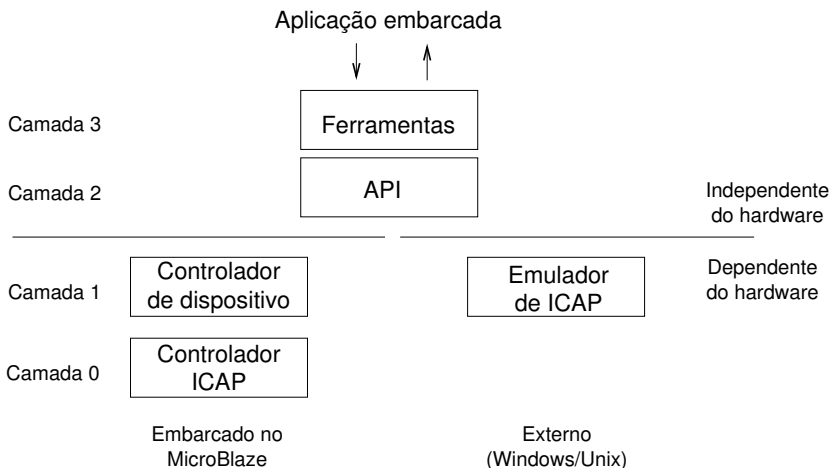

Figura 3.6: Estrutura de software do controlador de configurações descrito em [BLO03] – *O software é responsável pela interface com a aplicação embarcada e por controlar o processo de configuração, gerenciando o acesso a interface ICAP.*

- um monitor de condições de reconfiguração opcional;
- um armazenamento do estado do sistema (ou do estado de configurações);
- um carregador de configurações;
- um relocador de módulos opcional.

A partir dessas constatações pode-se imaginar como é a estrutura geral de um controlador de configurações, apresentada na Figura 3.7, ou seja, quais são seus componentes e principalmente a funcionalidade de cada um deles. Em seguida, pode-se propor uma estrutura de controlador e estudar quais são as melhores estratégias a serem abordadas e quais são viáveis. Com base nesses mesmos estudos pôde-se elaborar um quadro comparativo, como o apresentado na Tabela 3.1.

Tabela 3.1: Resumo dos controladores de configurações estudados – *Quadro comparativo dos modelos e implementações de controladores de configurações apresentados anteriormente nesse Capítulo. A última coluna da dessa Tabela apresenta as características do modelo RSCM aqui proposto, apresentado no Capítulo 4.*

	Modelos			*Implementações*		*Proposta*
Características	**Shirazi et al.**	**Burns et al.**	**Lysaght et al.**	**Curd**	**Blodget et al.**	**RSCM**
Hardware/software	Não aplicável	Não aplicável	Não aplicável	Software	Software	Hardware
Dispositivo alvo	XC6200	XC6200	XC6200	VirtexII-Pro	VirtexII	VirtexII
Escalonamento	Estático	Estático	Dinâmico	Nenhum	Nenhum	Estático
Preempção	Não	Não	Sim	Não	Não	Não*
Relocação	Não	Sim	Sim	Não	Não	Não*
Armazenamento de configurações	Sim	Sim	Sim	Sim	Não	Sim
Decodificação de configurações	Não	Não	Sim	Não	Não	Não
Localização do controlador	Não aplicável	Não aplicável	Não aplicável	Interno ao FPGA	Interno ao FPGA	Interno ao FPGA
Ano da publicação	1998	1997	1999	2003	2003	2003

*Implementação futura prevista.

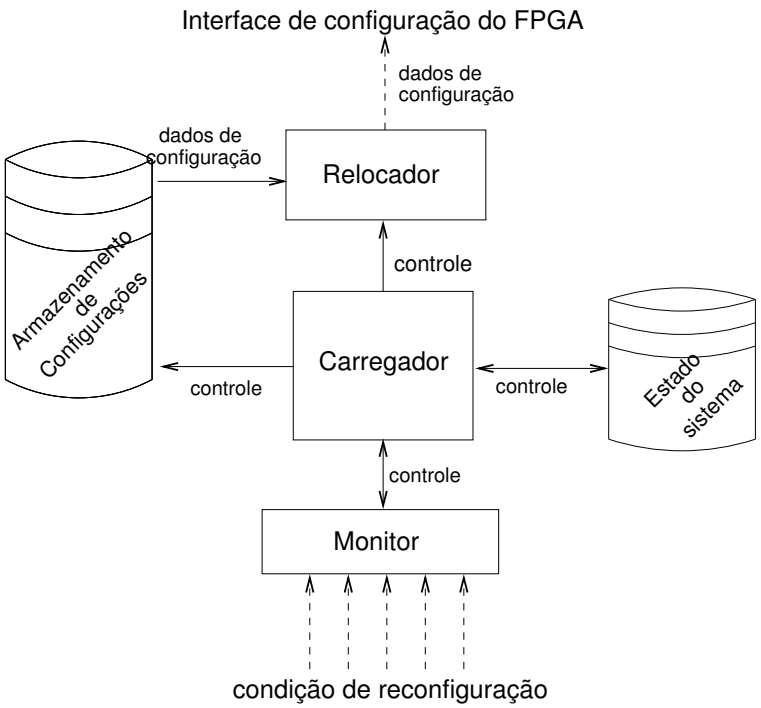

Figura 3.7: Proposta de estrutura geral de um controlador de configurações – *Elaborada de acordo com a observação dos modelos e implementações de controladores de configurações estudados.*

O modelo mais sofisticado prevê o uso de preempção além da possibilidade de manipular dados compactados ou encriptados. Ele ainda possui um escalonador, que nos outros modelos não faz parte do sistema e apenas fornece entradas para o controlador de configurações.

As três primeiras propostas apresentadas foram desenvolvidas visando sobretudo uma família de FPGAs descontinuada, a XC6200 da Xilinx [XIL97]. Os dois últimos modelos, no entanto, foram desenvolvidos para arquiteturas atualmente disponíveis. O primeiro deles para dispositivos VirtexII-Pro da Xilinx Inc. e o segundo para dispositivos da família VirtexII, do mesmo fabricante.

Nenhuma das implementações apresentadas segue algum dos modelos propostos. No presente trabalho será apresentado um sistema de controle de configuração que possui a maioria dos submódulos propostos nos modelos apresentados anteriormente. As implementações de Curt e Blodget et al. refletem sistemas controladores de configurações implementados parcial ou totalmente em software. A abordagem aqui adotada será implementar o sistema de controle de configurações em hardware. Estuda-se tal abordagem com o objetivo de obter informações relevantes para futuramente elaborar um comparativo entre o modelo aqui proposto, em hardware, e modelos desenvolvidos em software. Até o presente instante é impossível estimar de maneira adequada qual das abordagens é a melhor porque

nenhum sistema dessa natureza foi implementado inteiramente em hardware.

3.4 Gerenciamento de recursos em SDRs

Vistas algumas das principais propostas e implementações de controladores de configurações, algumas decisões importantes devem ser tomadas ainda no projeto do SDR. Estas influem diretamente na complexidade do controlador projetado e na sua eficiência. Nas Seções abaixo, aborda-se os seguintes temas:

- Estratégias de alocação em dispositivos reconfiguráveis;
- Particionamento de hardware;
- Políticas de escalonamento de configurações;
- Relocação de configurações

3.4.1 Estratégias de alocação em dispositivos reconfiguráveis

A arquitetura de dispositivos reconfiguráveis ainda é a maior fonte geradora de restrição para o projeto de um sistema controlador de configurações. Dependendo da forma como o dispositivo deve/pode ser manipulado o controlador poderá ou não apresentar estruturas mais complexas. Em [WAL03], apresenta-se quatro formas de modelar um dispositivo reconfigurável. São elas:

- bidimensional não-paginado;
- bidimensional paginado;
- unidimensional não-paginado;
- unidimensional paginado;

O modelo bidimensional não-paginado proporciona a maior flexibilidade, pois possibilita posicionar configurações ocupando uma superfície retangular em qualquer posição do dispositivo, como esboçado na Figura 3.8(a). A flexibilidade deste modelo dificulta as tarefas de escalonamento de posicionamento. Além disso, propicia a fragmentação externa quando uma configuração necessita de recursos que estão disponíveis no dispositivo mas não podem ser utilizados porque encontram-se dispersos.

Uma variação desse modelo propõe paginar o dispositivo em fatias com dimensões fixas, onde as configurações devem ser posicionadas como apresentado na Figura 3.8(b). A estas fatias dá-se o nome de blocos. Um bloco é agora a unidade mínima de configuração. Esta estratégia facilita o escalonamento e o posicionamento e causa fragmentação interna quando uma configuração necessita de menos recursos que os disponíveis em cada bloco de tamanho fixo.

Steinger [STE03], Walder e outros [WAL03] evidenciam uma outra desvantagem dos modelos bidimensionais: a dificuldade em prover meios de comunicação entre as configurações.

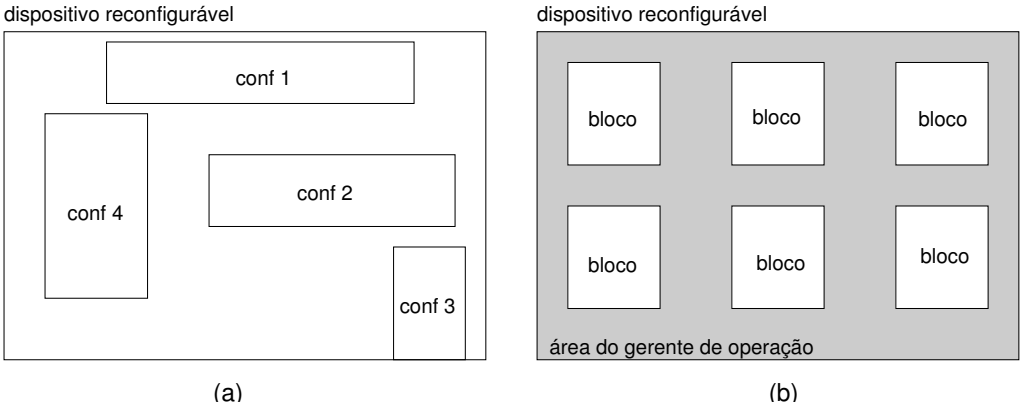

Figura 3.8: Modelos de alocação bidimensionais para dispositivos reconfiguráveis – *O modelo não-paginado (a) é o mais flexível mas proporciona fragmentação externa. O modelo paginado (b) é realista mas apresenta o problema de fragmentação interna. Nessa figura entenda conf1 como configuração 1*

O modelo unidimensional não-paginado restringe o posicionamento de configurações na direção vertical como esboçado na Figura 3.9(a). Nesse modelo, as configurações podem estar localizadas em qualquer porção horizontal, mas devem ocupar todo espaço vertical do dispositivo. Este modelo sofre com problemas de fragmentação interna e também externa.

O modelo unidimensional paginado restringe o posicionamento de configurações a blocos com ambas dimensões, vertical e horizontal, fixas como visto na Figura 3.9(a). Este modelo sofre com o problema de fragmentação interna, mas facilita muito as tarefas de escalonamento e posicionamento. Estes modelos são mais adequados para as restrições impostas pelas tecnologias hoje disponíveis no mercado (e.g. dispositivos VirtexII, que são paginados em quadros verticais).

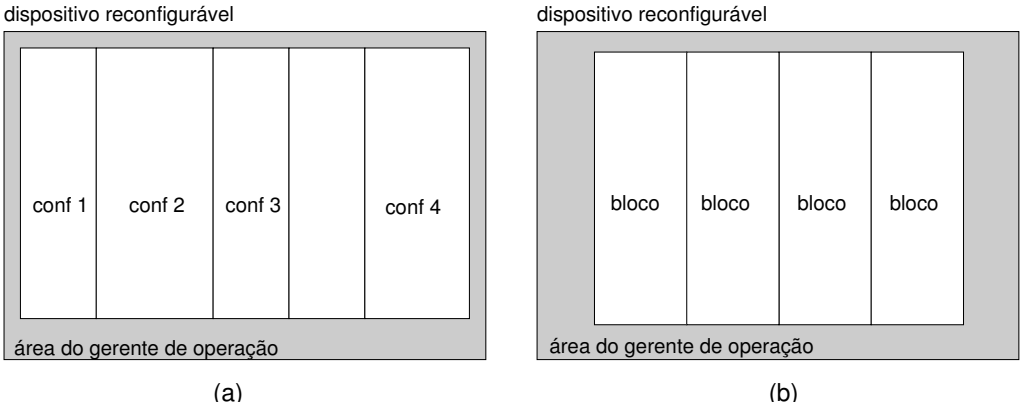

Figura 3.9: Modelos de alocação unidimensionais para dispositivos reconfiguráveis – *O modelo não-paginado (a) sofre com problemas de fragmentação interna e também externa. O modelo paginado (b) é o mais realista e apresenta o problema de fragmentação interna. Nessa figura entenda conf1 como configuração 1.*

O modelo mais próximo das características arquiteturais de dispositivos da família VirtexII da Xilinx é o unidimensional. Isto devido ao fato dos dispositivos desta família serem paginados em quadros verticais, como apresentado no Capítulo anterior.

3.4.2 Particionamento de hardware reconfigurável

Particionar um sistema consiste em separá-lo em *partes* menores (i.e. configurações parciais) mantendo inalterado o seu comportamento. O particionamento implica portanto em manter intacta a funcionalidade do sistema e sua interface com entidades externas. Dessa forma, o fato do sistema estar particionado é transparente a entidades externas.

O principal objetivo do particionamento é reduzir a complexidade do projeto de um sistema, onde cada parte pode ser tratada independentemente. Modernamente, no âmbito de sistemas configuráveis, outros objetivos adquirem maior importância, tal como solucionar o problema relacionado às restrições físicas do dispositivo configurável. Este pode não disponibilizar recursos suficientes para implementar completamente o sistema. Mais especificamente, em sistemas dinâmica e parcialmente reconfiguráveis, particiona-se o hardware para que através de um cronograma de configuração das partes geradas obtenha-se um ganho de desempenho chegando possivelmente a uma melhor utilização do dispositivo.

Segundo [ZHA00], o particionamento de sistemas dinâmica e parcialmente reconfiguráveis pode acontecer de duas formas:

- **Particionamento no domínio espacial**: divide o sistema em recursos estáticos de hardware;
- **Particionamento no domínio temporal**: divide o sistema dentro de segmentos de tempo exclusivos.

O particionamento espacial é a abordagem clássica em projetos de hardware em geral. O particionamento temporal explora a possibilidade de dividir o sistema no tempo, onde apenas parte deste é configurado a cada instante, enquanto outras partes são configuradas quando necessárias de acordo com a necessidade da aplicação.

Como citado em [MER98], obter a máxima vantagem de RTR na implementação de sistemas digitais propõe o estudo de problemas tal como o particionamento de hardware. De forma mais específica, são necessárias técnicas para "particionar" e, ainda, definir um cronograma de reconfiguração adequado, maximizando o desempenho do sistema dinâmica e parcialmente reconfigurável, considerando as restrições de área do dispositivo reconfigurável. Na mesma referência, evidencia-se a importância de particionar o sistema de forma que a comunicação entre as partes geradas seja minimizada. De outro modo, o desempenho do sistema pode ser comprometido em face do gargalo gerado pela comunicação entre seus componentes. O problema de comunicação agrava-se quando uma parte do sistema

residente no dispositivo necessita comunicar-se com outra parte ainda não configurada. Métodos de salvamento de contexto para comunicação são necessários neste caso.

3.4.3 Políticas de escalonamento de configurações

O conceito escalonamento aplica-se freqüentemente a assuntos relacionados a software, especificamente em sistemas operacionais. Segundo [TAN01], quando mais de um programa encontra-se apto a executar, o sistema operacional deve decidir qual deles irá fazê-lo. A parte do sistema operacional responsável por tomar esse tipo de decisão recebe o nome de *escalonador*. Cabe ao escalonador decidir, segundo uma política dada a ordem na qual os programas serão executados pelo processador. Mas não é tarefa do escalonador aplicar sua decisão. Desta maneira, um mecanismo de suporte deve ser provido para, por exemplo, carregar o programa escolhido no processador.

Por analogia a software, em SDRs configurações correspondem a programas e o dispositivo reconfigurável corresponde ao processador. O mecanismo que aplica as decisões tomadas pelo escalonador é o controlador de configurações.

Para escopo de SDRs, Vasilko [VAS95] define o problema de escalonamento como: "*Dado um conjunto de configurações e a área total do dispositivo reconfigurável, encontrar o melhor escalonamento que obedeça a restrições de precedência (ou prioridade) e que o somatório das áreas de cada uma das configurações fisicamente presentes no dispositivo não seja maior que a área total.*"

A estratégia de interromper a execução de uma configuração (processo) quando uma outra de maior prioridade encontra-se apta a executar recebe o nome de *preempção*. O escalonador pode aplicar técnicas de preempção, dependendo da política adotada. Seu emprego, no entanto, exige a presença de entidades responsáveis por gerenciar o estado atual de execução da configuração a ser preemptada. Esta entidade responsabiliza-se por armazenar o estado da configuração para que este seja 'retomado' quando a configuração retornar ao dispositivo. Como citado em [WAL03], para preemptar uma configuração executando em um FPGA é necessário executar uma tarefa de readback e salvar o estado da configuração para que, retornando ao dispositivo, esta continue a operar do ponto onde foi interrompida.

O escalonamento de configurações pode ser dinâmico ou estático. Um sistema pode ter seu cronograma de escalonamento definido em tempo de projeto, sendo portanto fixo. Este modelo recebe o nome de escalonamento *estático*. De outra forma, quando as decisões são tomadas em tempo de execução o escalonamento é dito *dinâmico*. Este último modelo indica uma nova área de pesquisas ainda inexplorada. O fato de uma configuração residente no dispositivo realizar operações que venham a interferir no cronograma de configurações apresenta-se como a realização de saltos em software. Assim sendo, a técnica conhecida como *predição de saltos* pode também ser aqui aplicada, como citado em [VIS03]. Uma

denominação para esta técnica pode ser "*predição de configuração*".

A tarefa de escalonar configurações é, portanto, crucial para o desempenho de SDRs. Estratégias simplistas podem acarretar, por exemplo, redução da densidade funcional do sistema. Embora seja notória a importância de métodos de escalonamento adequados a SDRs, a maioria das pesquisas nesta área objetivam sistemas reconfiguráveis baseados em múltiplos contextos [MAE01a, MAE01]. A seguir, apresentam-se algumas propostas de métodos de escalonamento.

Propostas de Walder et al

Em [WAL03], Walder e Platzner apresentam quatro modelos de escalonamento. Estes são agrupados em duas classes. São elas:

- **Sem preempção:** onde toda configuração executa até acabar sua tarefa.
  - **FCFS** - em inglês *First Come First Served*, a configuração executa de acordo com a ordem de chegada. Comporta-se tal como uma fila, onde a primeira a chegar é a primeiro a executar;
  - **SJF** - em inglês *Shortest Job First*, onde o escalonador toma a decisão de acordo com o tempo de execução, possivelmente estimado, de cada tarefa. A configuração que necessita ocupar o dispositivo por um tempo menor é executada primeiro.
- **Com preempção:** quando uma configuração executando pode ser interrompida e removida (preemptada) do dispositivo para dar lugar a outra de maior prioridade.
  - **SRPT** - em inglês *Shortest Remaining Processing Time*, onde escalona-se para execução a configuração que possivelmente terminará sua tarefa antes, ou seja que necessita de menos tempo para executar;
  - **EDF** - em inglês *Earliest Deadline First*, onde escalona-se para execução a configuração que necessita terminar sua execução antes, ou seja, que possui uma maior necessidade do recurso.

Os métodos sem preempção podem causar postergação indefinida. Em FCFS uma configuração pode tomar um tempo arbitrariamente grande para executar e as demais configurações podem ser obrigadas a esperar indefinidamente. Em inglês, este fenômeno recebe o nome *starvation*. O método SJF também pode causar a postergação indefinida, porque as configurações que necessitam de um tempo maior para executar podem nunca chegar a ser escalonadas. Isto pode ocorrer quando diversas configurações que necessitam de um tempo menor requerem recurso para executar, ou seja, sempre existe uma outra configuração que necessita de menos tempo esperando para executar.

No método SRPT, as configurações que necessitam de um maior tempo para executar podem nunca ser escalonadas porque configurações mais rápidas sempre poderão ganhar a posse do dispositivo, em detrimento das primeiras.

Na mesma referência [WAL03] são apresentados os possíveis estados de uma configuração em um sistema que emprega técnicas de escalonamento com preempção. Na Figura 3.10 apresenta-se um grafo ilustrando os possíveis estados e transições de uma configuração em sistemas com preempção.

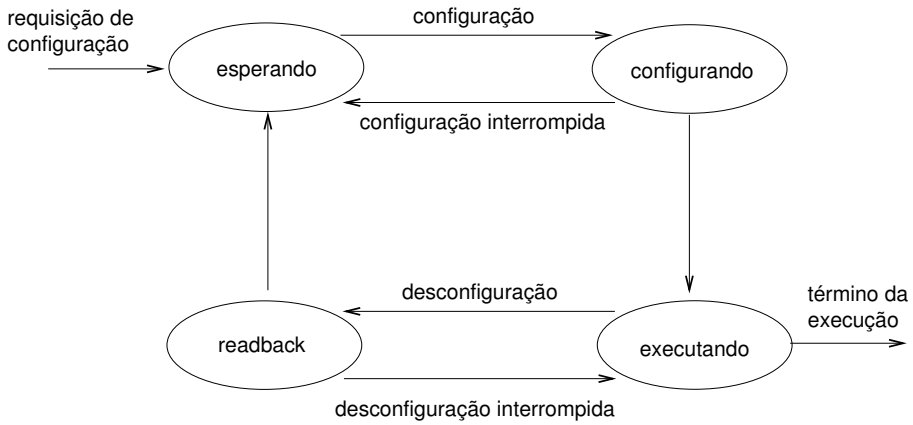

Figura 3.10: Diagrama de transição de estados de uma configuração para escalonamento com preempção proposto em [WAL03] – *Quando uma configuração chega, ela espera (**esperando**) até ser selecionada para ser configurada. Então, quando escalonada para execução passa ao estado **configurando** e, se o processo não for interrompido, ao estado **executando**. Acontecendo uma preempção da tarefa executando ela retorna ao estado esperando. Antes disto, porém, deve acontecer uma operação de readback que salva seu estado, ilustrado pelo estado **readback**.*

Em [STE03], Steiger e outros apresentam duas heurísticas de escalonamento sem preempção denominadas *horizon* e *stuffing*.

A técnica *horizon* utiliza duas listas para implementar o escalonamento. A primeira delas, denominada *horizonte de escalonamento*, contém informação sobre os intervalos vazios, onde as configurações podem ser inseridas. Os intervalos são representados por três valores: um que indica o tempo quando este recurso estará livre e os outros dois que indicam a largura da área. A estrutura $[2, 10]@3$ indica que no tempo *3* a área entre *2* e *10* estará livre. A segunda lista, chamada *lista de reserva*, apresenta a seqüência de escalonamento informando em qual tempo, deve ser inserida a configuração e qual a sua posição. A sua representação utiliza três valores. A estrutura $3(1, 5)$ indica que a configuração *C3* deve ser inserida na posição *1* no tempo *5*. Quando uma configuração chega, o escalonador percorre a lista de intervalos e verifica quando e onde a configuração pode ser inserida. De acordo com o resultado da pesquisa, atualiza estas duas listas.

A técnica *stuffing* escalona configurações em retângulos livres de maneira arbitrária. Ela também utiliza duas listas: uma *lista de espaço livre* e uma *lista de reserva*. O escalonador stuffing percorre as listas, simulando todas as futuras alocações, emulando seus términos e unindo os espaços livres, ou seja, aplica a técnica do melhor lugar (*best fit*). Um comparativo dos resultados do escalonamento da configuração sete (*C7*) é apresentado na Figura 3.11.

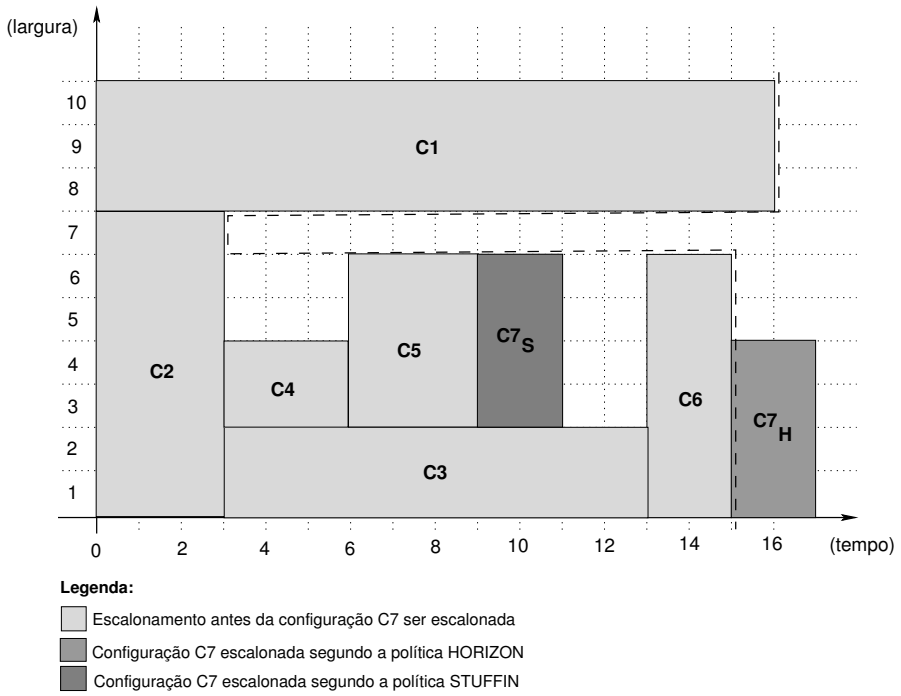

Figura 3.11: Gráfico representando o resultado do escalonamento da configuração *C7* segundo as políticas *HORIZON* e *STUFFING* apresentadas em [STE03] – *As duas técnicas usam duas listas com conteúdos semelhantes, uma com informações sobre o estado de alocação do dispositivo e outra com informações de escalonamento sendo gerado. A diferença entre essas técnicas é como e quando as listas são atualizadas.* HORIZON *atualiza apenas quando uma configuração chega, ao passo que* STUFFING *atualiza de tempos em tempos, quando uma configuração é inserida e quando é retirada. Em virtude disso, antes de inserir a configuração C7 a técnica HORIZON tem como área livre apenas aquela a esquerda da linha pontilhada. Pode-se notar nessa Figura que o dispositivo empregado possui largura 10.*

Na Figura 3.11 pode-se perceber que o resultado apresentado pela segunda técnica aumenta a densidade funcional do sistema, ou seja, melhora o desempenho do mesmo. O preço pago por este desempenho é a complexidade para simular os futuros términos das configurações. As duas técnicas usam duas listas com conteúdos semelhantes, a diferença está em como e quando elas são atualizadas. *Horizon* atualiza apenas quando uma configuração chega ao passo que *stuffing* atualiza de tempos em tempos, quando uma configuração é inserida e quando é retirada.

As duas técnicas tratadas visam escalonamento de tarefas sem preempção, tal qual o modelo RSCM proposto. Os autores dizem que o tratamento de grafos de dependência é uma tarefa a ser tratada. Os mesmos já são tratados no modelo RSCM, como será visto no Capítulo 4.

3.4.4 Relocação de configurações

Em [COM00], Compton et al. definem *relocação* como a habilidade de determinar em tempo de execução o posicionamento de uma configuração no dispositivo configurável. Sua aplicação exige métodos para modificar/adequar uma dada configuração, projetada para uma localização específica no dispositivo, de forma que a mesma possa ser inserida/configurada em uma outra localização.

Através de relocação, pode-se reduzir substancialmente o tamanho da memória necessária para armazenar as configurações. Devido as técnicas de relocação, pode não ser necessário armazenar diversas configurações com o mesmo comportamento desenvolvidas para localizações diferentes no dispositivo reconfigurável.

Segundo [COM00], uma *colisão* ocorre quando uma configuração a ser realizada foi projetada para ocupar uma localização que sobrepõe uma outra já residente no dispositivo. Quando esse evento ocorre, apenas uma das configurações pode estar residente no dispositivo num dado instante. A técnica de relocação resolve o problema de colisão. Quando duas configurações colidem, pelo menos uma delas deve ser relocada. Pode ocorrer das duas necessitarem ser relocadas. Na Figura 3.12 apresenta-se como relocação pode resolver um problema de colisão.

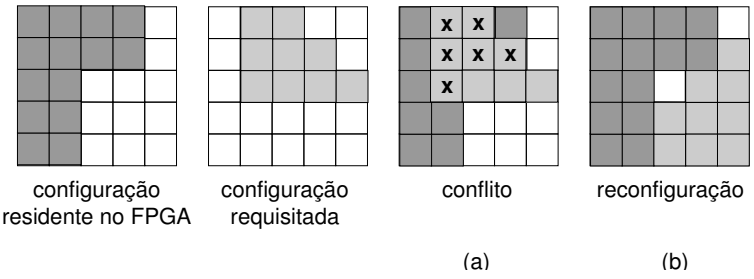

Figura 3.12: O problema de colisão de configurações [COM00] – *Quando uma configuração requisitada colide com uma residente no dispositivo dois eventos podem ocorrer: (a) sem estratégia de relocação a configuração não pode ser realizada; (b) a configuração requisitada é relocada para que a requisitada seja realizada.*

Em [COM02a], são apresentados três movimentos distintos passíveis de serem realizados sobre configurações. São eles: rotação de 90^0, *flip* horizontal e vertical. Combinando

estes três movimentos pode-se realizar sete manipulações diferentes como apresentado na Figura 3.13.

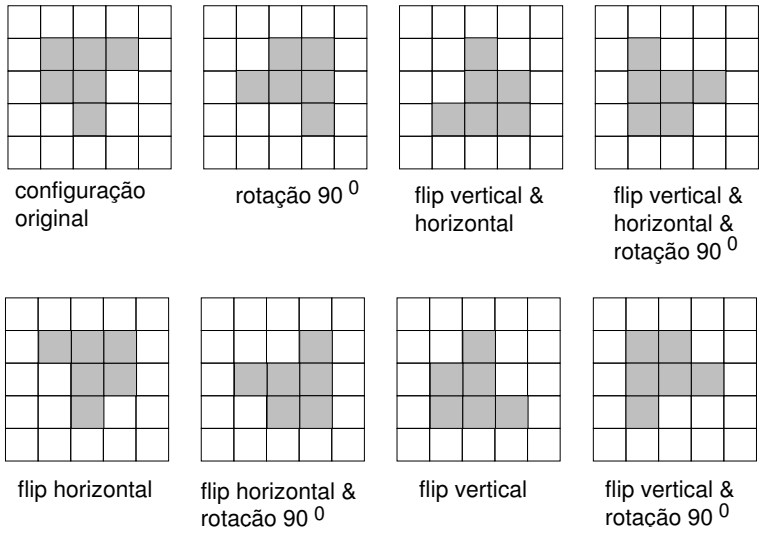

Figura 3.13: Sete manipulações primárias de uma configuração segundo [COM02a] – *Estes movimentos são usados para aplicar relocação de configurações.*

As manipulações apresentadas na Figura 3.13 não refletem a realidade dos dispositivos reconfiguráveis atualmente comercializados porque pressupõem a existência de dispositivos manipuláveis de maneira bi-dimensional. Como já foi citado, os dispositivos hoje comercializados são particionados em colunas e uma operação de configuração exige a configuração atômica de pelo menos uma coluna de elementos. Atualmente, com base nos estudos realizados, pode-se admitir que a relocação de configurações poderia ser realizada apenas sob a forma de deslocamentos horizontais.

Em [MES02], apresenta-se estudos sobre a relocação de configurações específicos para dispositivos Virtex da Xilinx. No trabalho de Mesquita foram desenvolvidas ferramentas que possibilitam a edição de arquivos e configuração bem como a geração de arquivos de configuração parciais, citadas no Capítulo anterior. Em [STE03], Steiner e outros dizem ter solucionado o problema de relocação de configurações para dispositivos da família Virtex da Xilinx. Maiores detalhes sobre como este problema deve ser tratado não são documentados pelos autores. Em virtude das restrições impostas pelo tempo de desenvolvimento desse trabalho, maiores detalhamentos sobre relocação de configurações não serão aqui abordados.

Capítulo 4

RSCM - Uma proposta de controlador de configurações

Como foi citado em [VIL97] e reforçado em [SAR01]: "*Uma arquitetura de hardware que tem a possibilidade de reconfigurar a si mesma dinamicamente à medida que uma tarefa é executada, refinando a sua própria configuração para obter um melhor desempenho é a forma mais desafiadora e potencialmente mais poderosa de um sistema computacional configurável*".

A tarefa de reconfigurar partes de um dispositivo necessita ser controlada por uma entidade. Esta entidade, denominada aqui *controlador de configurações*, deve receber informações de uma entidade escalonadora de configurações, além de informações contendo o estado atual do dispositivo e, a partir delas, gerenciar de maneira adequada o processo de reconfiguração.

O principal objetivo do presente trabalho é propor e desenvolver um hardware controlador de configurações, que recebe o nome de *Reconfigurable System Configuration Manager* (RSCM). Esta proposta limita-se ao desenvolvimento de uma implementação puramente em hardware, onde o dispositivo configurável possui internamente toda, ou a maior parte da lógica de controle de configurações. Esta limitação é imposta pelo tempo reduzido para que se possa implementar a solução também em software, soluções mistas e realizar a comparação de compromissos.

Parte da infra-estrutura de reconfiguração parcial, apresentada na Seção 1.4, é suposta como funcional. Em particular assume-se como resolvido o problema de gerar configurações parciais. Para maiores detalhes o leitor pode referir-se aos trabalhos propostos em [BRI03] e [OST03], correspondentes ao suporte para desenvolvimento de configurações parciais e à padronização da interface de comunicação entre diferentes módulos, respectivamente. Assume-se também que a arquitetura alvo compõe-se de dispositivos da família Virtex-II [XIL02c] fabricados pela empresa Xilinx, como anteriormente citado.

Pressupõe-se o uso de uma arquitetura alvo onde o dispositivo é dividido em um certo número de fatias iguais, cada uma podendo conter exatamente um módulo de hardware reconfigurável e uma ou mais fatias contendo hardware fixo não reconfigurável. Isto é suposto para que não se necessite tratar questões de fragmentação de área no dispositivo reconfigurável. Por outro lado, trata-se o problema relacionado a relocação de configurações, para que essas sejam "configuráveis" em qualquer fatia do dispositivo, sendo a decisão de qual fatia será ocupada tomada em tempo de execução. Par isso, não foi desenvolvido um módulo relocador, mas como a decisão é tomada em tempo de execução, então diversas cópias de cada arquivo de configuração devem ser armazenadas. O modelo de particionamento de dispositivo segue o modelo unidimensional particionado apresentado na Figura 3.9(b) da Subseção 3.4.1.

As implementações de SDRs contempladas no presente trabalho limitam-se àquelas em que a comunicação ocorre apenas entre duas configurações ativas, denominada aqui *comunicação em linha*. A comunicação entre uma configuração ativa e outra configuração num estado diferente (Figura 1.3) recebe o nome de *configuração postergada*. Esse tipo de configuração pode, contudo sempre ser obtida no contexto de aplicações específicas utilizando, por exemplo, acesso à memória externa compartilhada por duas configurações. A limitação imposta aqui é não prover, na infra-estrutura de suporte a SDRs proposta, a automatização do intercâmbio de informações entre módulos reconfiguráveis que pretendam usar comunicação postergada.

Na Seção 4.1 do presente Capítulo apresenta-se a estrutura do sistema proposto. Na Seção 4.2 apresenta-se um exemplo ilustrativo da operação do sistema RSCM.

4.1 Estrutura do RSCM

O diagrama de blocos do sistema aqui proposto e implementado aparece na Figura 4.1. O sistema é composto por seis módulos principais. Existe uma Memória de Configurações (MC) que armazena as configurações do sistema. Segue-se o módulo Autoconfigurador (AC), responsável por comandar a tarefa de controlar a reconfiguração do dispositivo propriamente dita. O módulo Interface de Configuração (IC) responsabiliza-se pela interface entre a porta de configuração do dispositivo reconfigurável (porta para onde os dados de configuração devem ser enviados) e o sistema RSCM. O módulo responsável por monitorar eventos (sinalizações indicam a necessidade de um nova tarefa de configuração) disparados por configurações ativas é o Monitor de reconfiguração (MR). O Escalonador de Configurações (EC) determina qual a próxima configuração a ser reconfigurada. Para gerenciar a operação e coordenar a comunicação entre os demais módulos do sistema, existe um módulo chamado Controle Central de Configurações (CCC).

A seguir apresentam-se mais detalhes sobre as atividades realizadas pelos submódulos do RSCM.

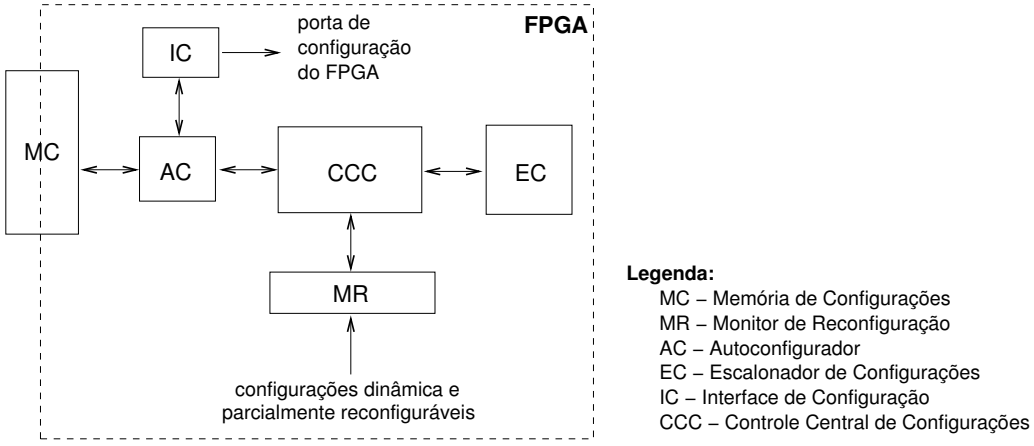

Figura 4.1: Diagrama de blocos do controlador RSCM proposto – *O modelo RSCM proposto é composto por uma Memória de Configurações que armazena as configurações; um módulo Autoconfigurador responsável pela tarefa de configurar o dispositivo propriamente dita; Um módulo de Interface de Configuração responsável pela interface entre a máquina de configuração do dispositivo configurável e o sistema RSCM; um módulo responsável por monitorar eventos disparados por configurações ativas, denominado Monitor; um módulo responsável por manter as informações que refletem o Escalonamento do sistema e; um módulo encarregado de gerenciar a operação e a comunicação entre os demais módulos do sistema, denominado Controle Central de Configurações ou simplesmente CCC.*

4.1.1 O Monitor de Reconfiguração

O *Monitor de reconfiguração* (MR) é o módulo do sistema RSCM responsável por detectar/reconhecer situações nas quais devem ser realizadas reconfigurações, denominadas *condições de reconfiguração*. Toda a operação do sistema RSCM inicia-se a partir de uma condição de reconfiguração que identifica quando o sistema necessita ser reconfigurado.

Uma implementação do Monitor de reconfiguração deve identificar quando uma dada configuração termina sua execução (condição de reconfiguração) e notificar o Controlador Central de Configurações para que este tome as devidas providências.

4.1.2 O Controle Central de Configurações

O *Controle Central de Configurações* (CCC) é responsável por gerenciar o fluxo de controle entre os demais módulos do sistema RSCM. O CCC recebe notificações de condições de reconfigurações provenientes do Monitor de reconfiguração. Em seguida, requisita tarefa de escalonamento ao Escalonador de Configurações e requisita tarefas de configuração ao módulo Autoconfigurador.

O Controle Central de Configurações é composto por três partes: Sub-módulos de Lógica de Controle, Escalonamento e Configuração. Na Figura 4.2 apresenta-se a estrutura do módulo CCC.

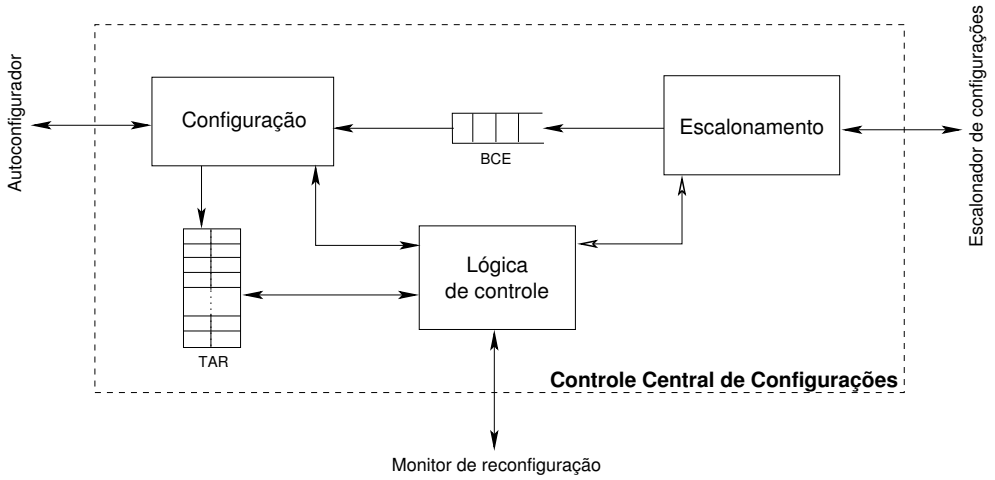

Figura 4.2: Controle Central de Configurações do sistema RSCM – *Módulo do sistema RSCM composto por três partes: Lógica de Controle, Escalonamento e Configuração. A Lógica de Controle mantém informações sobre a alocação do dispositivo e configurações escalonadas. O Escalonamento recebe as notificações de condições de reconfiguração e requisita escalonamentos. O módulo Configuração requisita reconfigurações.*

O sub-módulo *Lógica de Controle* é responsável por manter informações sobre o estado de alocação do dispositivo, ou seja, quais configurações encontram-se configuradas e ainda em qual das fatias. Também mantém informação sobre o diagrama de configurações escalonadas. Além disso, este módulo controla a operação dos outros dois submódulos através de sinais de controle. Um exemplo do estado de alocação do dispositivo, representado através da *Tabela de Alocação de Recursos* (TAR), encontra-se na Tabela 4.1.

O sub-módulo *Escalonamento* é responsável por:

- Escalonador de Configurações (EC) receber as notificações de condição de reconfiguração;
- alterar a estrutura de alocação do dispositivo;
- requisitar serviços ao Escalonador de Configurações e armazenar o código de configurações escalonadas no *Buffer de Configurações Escalonadas* (BCE).

O sub-módulo *Configuração* responsabiliza-se por requisitar ao Autoconfigurador a configuração dos bitstreams escalonados de acordo com BCE. Após a tarefa de configuração esse submódulo atualiza a TAR. Na Tabela 4.1 apresenta-se a um exemplo de uma Tabela de Alocação de Recursos que contém informações sobre qual bitstream parcial encontra-se configurado em cada uma das fatias reconfiguráveis do dispositivo.

Tabela 4.1: Estrutura da Tabela de Alocação de Recursos (TAR) – *Esta tabela representa um dispositivo particionado em cinco fatias. Na fatia zero encontra-se configurada a configuração C3; na fatia um encontra-se configurada a configuração C1 e assim por diante.*

Número da fatia	***Configuração***
0	C3
1	C1
2	C4
3	C2
4	C6

4.1.3 O Escalonador de Configurações

O *Escalonador de Configurações* (EC) é o módulo responsável por determinar dinamicamente o cronograma de execução das configurações. Tal módulo recebe requisições de serviço vindas do Controle Central de Configurações. O escalonador possui uma estrutura de dados que contém informações sobre as dependências entre as configurações, denominada *Tabela de Dependência e Descritores* (TDD), como a apresentada na Tabela 4.2, onde *Config* indica o identificador da configuração; *Npredec* indica o número de configurações predecessoras da configuração; e *Suc N* é a sucessora N da configuração. A Tabela 4.2 representa o grafo de dependência apresentado na Figura 4.3.

O número de linhas da tabela depende do número de configurações parciais do sistema. Este valor consiste em uma parametrização do sistema RSCM. Dependendo do número de configurações possíveis, do número máximo de predecessoras e do número máximo de sucessoras, esta tabela terá dimensões diferentes.

Tabela 4.2: Estrutura da Tabela de Dependência e Descritores (TDD) – *Tabela que armazena informações referentes ao escalonamento de configurações, gerada a partir do grafo de dependência apresentado na Figura 4.3.*

Config.	***Npredec.***	***Suc. 1***	***Suc.*** 2
0	2	1	2
1	1	5	4
2	1	4	3
3	1	0	-
4	2	0	-
5	1	-	-

O grafo da Figura 4.3 representa um cronograma de execução de um sistema dinâmica e

parcialmente reconfigurável composto por seis configurações. As arestas que chegam a um determinado nodo recebem o nome de *predecessoras* desse nodo. Uma dada configuração só pode iniciar a execução, ou seja, pode ser escalonada para execução, quando todos os seus predecessores houverem concluído a execução, salvo na inicialização do sistema, quando a configuração zero inicia independentemente de suas predecessoras. As arestas que saem de um determinado nodo são ditas *sucessoras* desse nodo. Quando uma configuração termina sua execução ela deve notificar suas sucessoras, diminuindo o número de predecessoras de cada uma das suas sucessoras. Para um melhor entendimento, imagine que o nodo 1 termine de executar. Em virtude disso, as configurações 5 e 4 não necessitam mais aguardar o término de 1. A configuração 5 pode ser escalonada enquanto que a configuração 4 só poderá ser escalonada após o término da execução da configuração.

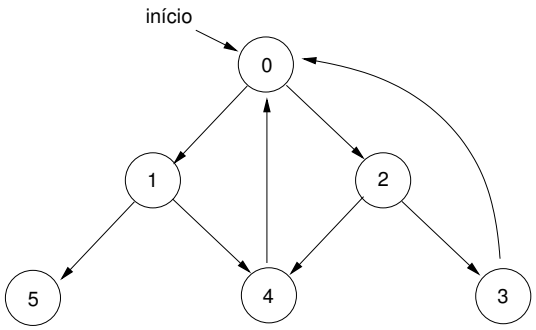

Figura 4.3: Exemplo de grafo de escalonamento de configurações – *Grafo de dependência de configurações utilizado para criar a TDD da Tabela 4.2.*

4.1.4 A Memória de Configurações

A função da *Memória de Configurações* (MC) é armazenar todos os bitstreams parciais utilizados no sistema reconfigurável. A princípio, MC pode ser interna ou externa ao dispositivo. Entretanto, conforme cresce a complexidade dos sistemas desenvolvidos, o número de configurações parciais necessárias cresce e muita memória torna-se necessária para armazenamento de configurações. Como arquivos de configurações parciais para dispositivos da família Virtex-II, alvo desse desenvolvimento, tem em torno de 40kbits, torna-se inviável implementar uma Memória de Configurações totalmente interna ao dispositivo. Na Seção 5.6 serão esclarecidos os motivos para escolha da localização da MC. Em caso de dispositivos de outras famílias e fabricantes, com maiores recursos internos de memória, pode-se pensar em uma pequena memória interna ao dispositivo com as configurações mais usadas, tal qual uma cache de configurações. Tal possibilidade não será utilizada aqui.

Para acessar a interface com a memória de uma maneira mais adequada e simplificada é necessário desenvolver um módulo *Controlador de acesso à memória*. Tal módulo empacota a memória de configurações, tornando transparente o protocolo de comunicação específico

com a mesma. Este módulo apresenta ao Autoconfigurador uma interface simplificada. De acordo com a memória a ser utilizada, é necessário desenvolver um novo controlador, adequado à nova interface física.

Na Figura 4.4 apresenta-se a estrutura do módulo MC, composto por uma memória acessada por meio de um *Controlador de acesso à memória*. O submódulo *Inicializador de Memória*, como o próprio nome indica é utilizado para inicializar a memória de configurações através da interface serial a partir de um computador hospedeiro. O hospedeiro acessa a Memória de Configurações através do submódulo denominado *Interface serial*. Toda parte de controle de acesso à memória encontra-se internamente ao dispositivo reconfigurável, enquanto a memória em si localiza-se externamente os dispositivo.

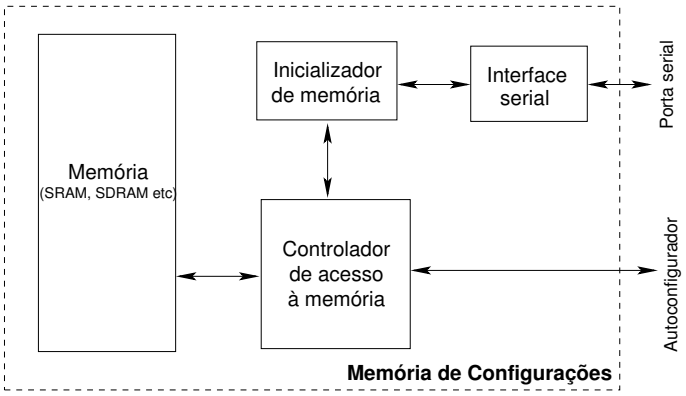

Figura 4.4: Memória de Configurações do sistema RSCM – *Memória que armazena as configurações utilizadas no sistema. O módulo controlador de acesso á memória apresenta ao Autoconfigurador uma interface simplificada. O Inicializador de memória inicializa a memória de configurações através da Interface serial.*

4.1.5 O Autoconfigurador

Como se sabe, o tempo de reconfiguração é um fator importante que reflete diretamente no desempenho de um SDR [LYS97]. Em virtude disso, a implementação do módulo Autoconfigurador, principal componente do sistema RSCM, deve se ater a estratégias que minimizem o tempo gasto para realizar a reconfiguração parcial do dispositivo. Caso contrário, a implementação do SDR poderá ser comprometida.

O *Autoconfigurador* é o módulo responsável por realizar a tarefa de configuração propriamente dita. Ele possui interface com a Memória de Configurações, Interface de Configuração e Controle Central de Configurações conforme ilustrado na Figura 4.1.

O Autoconfigurador recebe e interpreta requisições de reconfigurações provenientes do

Controle Central de Configurações. De acordo com as informações recebidas junto com a requisição, AC descobre em quais posições de memória deve ler os dados do arquivo de configuração. Logo após ler o arquivo de configuração da memória, AC envia os dados à Interface de Configuração, juntamente com sinais de controle.

Como apresentado na Figura 4.5, o módulo AC é composto por três sub-módulos. O primeiro deles, *Lógica de Controle*, é responsável por receber a requisição de reconfiguração, selecionar a área de memória e controlar a operação dos outros dois módulos. O segundo submódulo, *Leitor de configurações*, é responsável por ler da memória de configurações o arquivo de configuração selecionado e armazená-lo, parcialmente, em um buffer interno. O último sub-módulo, *Configurador*, lê do buffer os dados de configuração e envia-os ao módulo Interface de Configuração, descrito na próxima Seção.

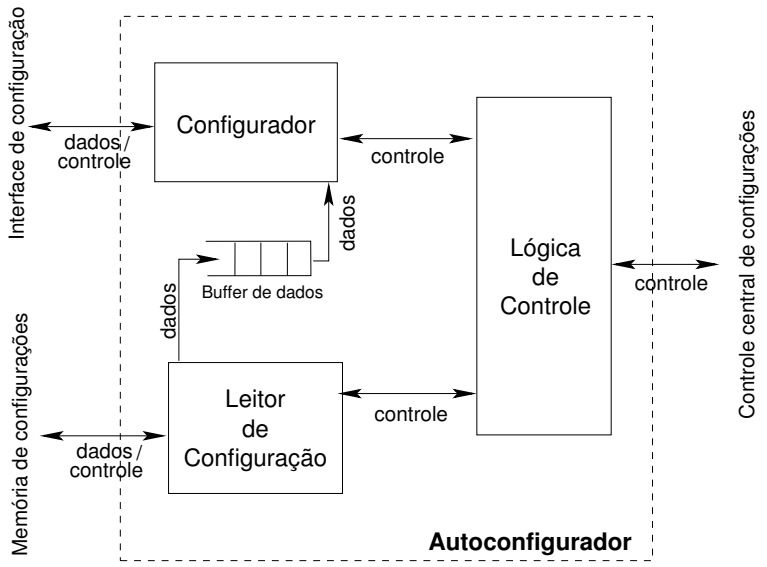

Figura 4.5: Autoconfigurador do sistema RSCM – *O Autoconfigurador recebe e interpreta requisições de reconfigurações provenientes do Controle Central de Configurações; Lê o arquivo de configuração da Memória de Configurações e o envia à Interface de Configuração. A Lógica de Controle recebe a requisição de reconfiguração. O Leitor de configurações lê a memória de configurações. O Configurador envia dados de configurações à Interface de Configuração.*

4.1.6 A Interface de Configuração

A *Interface de Configuração* é o módulo responsável por prover a interface com os pinos de configuração do dispositivo. Em dispositivos que provêem acesso interno aos pinos de configuração, tais quais os da família VirtexII e VirtexII-Pro da Xilinx, esse módulo pode ser implementado internamente ao dispositivo, possibilitando assim tempos

de reconfiguração mais eficientes. A quantificação de tempos de configuração será tratada no próximo Capítulo. No caso dos outros dispositivos, que não provêem acesso interno aos seus pinos de configuração, torna-se necessário desenvolver parte da lógica de IC externa ao dispositivo. Nesse caso, uma implementação razoável seria composta por fios externos ao dispositivo que transmitem os dados gerados internamente (ou lidos da memória) para os pinos de configuração do dispositivo acessados externamente.

O módulo de Interface de Configuração possui uma interface com o Autoconfigurador, de onde recebe os dados de configuração e sinais de controle; e uma interface com os pinos de configuração do dispositivo.

4.2 Operação do sistema RSCM

Para um melhor entendimento do sistema proposto nesse Capítulo, apresenta-se aqui, um esboço do comportamento de tal sistema representado desde uma condição de reconfiguração até a atualização da TAR (i.e. Tabela de Alocação de Recursos) do sistema.

A Figura 4.6 ilustra a execução dos seguintes passos:

Passo 1: um sistema dinâmica e parcialmente reconfigurável encontra-se executando. Num dado instante, todas as fatias estão ocupadas com configurações executando trabalho útil;

Passo 2: a **configuração 3**, que está executando na **segunda fatia**, notifica que terminou sua execução, disparando assim uma *condição de reconfiguração*;

Passo 3: o *Monitor de Reconfiguração* reconhece a *condição de reconfiguração* e notifica o término da execução da **configuração 3** ao *Controle Central de Configurações*;

Passo 4: o *Controle Central de Configurações* recebe a sinalização do *Monitor de Reconfiguração*, atualiza a *tabela de alocação de recursos* (TAR) e solicita ao módulo *Escalonador de Configurações* seus serviços;

Passo 5: o *Escalonador de Configurações* percorre a *tabela de dependência e descritores* (TDD) procurando por configurações aptas a serem configuradas e envia seus códigos, caso existam, ao *Controle Central de Configurações*;

Passo 6: o *Controle Central de Configurações* solicita ao *Autoconfigurador* a (re)configuração parcial das configurações escalonadas pelo *Escalonador de Configurações*;

Passo 7: o *Autoconfigurador* lê os dados de configuração da *Memória de Configurações* e os envia à *Interface de Configuração*, realizando, assim, a configuração do módulo escalonado;

Passo 8: o *Autoconfigurador* acaba a reconfiguração e o *Controle Central de Configurações* atualiza a *tabela de alocação de recursos*. O sistema opera, agora, com a **nova configuração** configurada na fatia onde se encontrava configurada a configuração 3, nesse caso, a **segunda fatia**.

No próximo Capítulo trata-se detalhadamente a implementação e validação de cada um dos módulos do sistema RSCM. Dessa forma, apresentam-se as opções existentes para implementar cada um dos módulos e os motivos para implementá-los da forma escolhida.

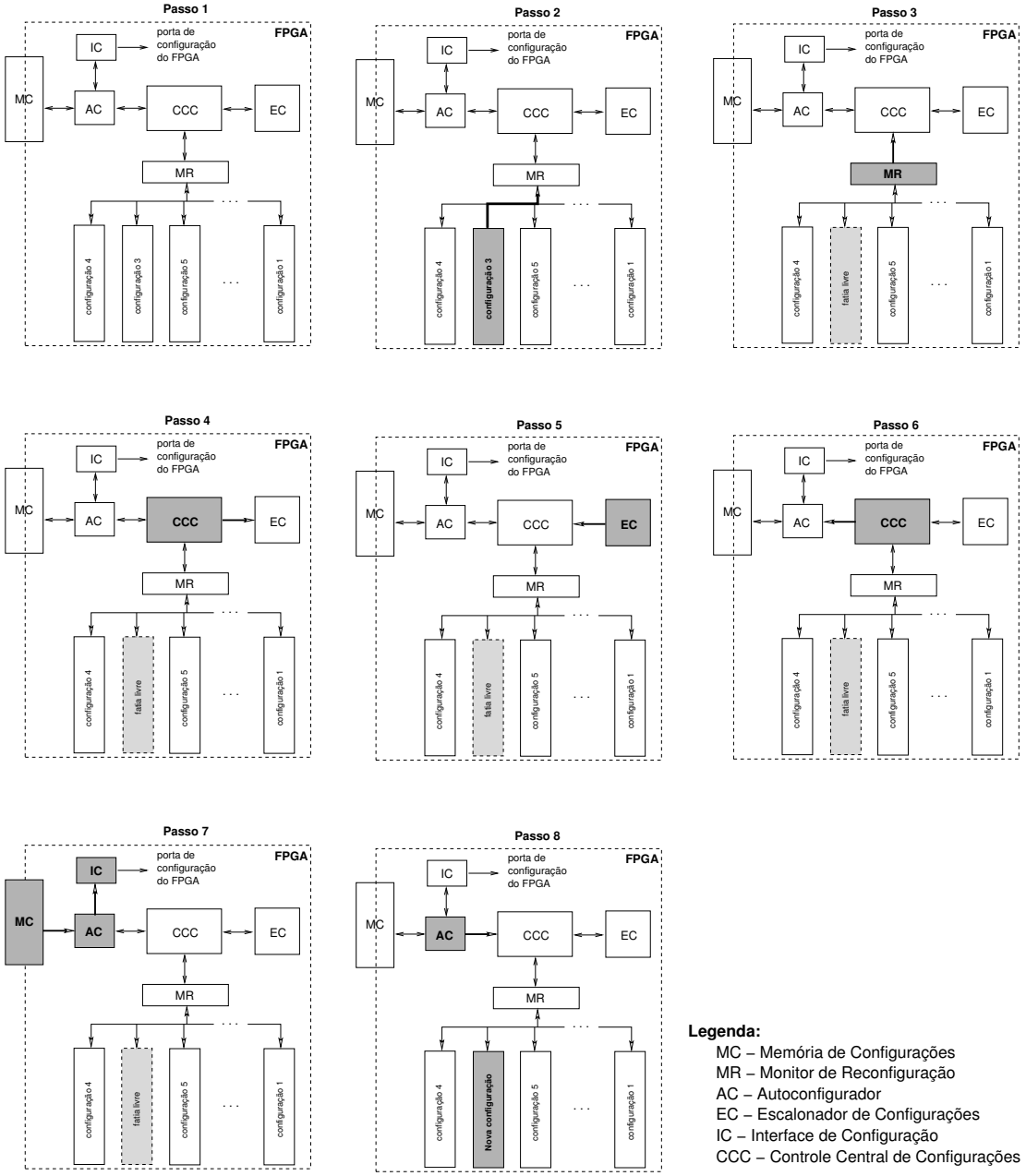

Figura 4.6: Exemplo de operação do sistema RSCM – *A operação do sistema RSCM inicia-se por uma condição de reconfiguração disparada pelo sistema reconfigurável. A partir daí os módulos do sistema RSCM executam suas tarefas, até que uma nova configuração parcial é reconfigurada no fatia liberada pela configuração que acabou de executar.*

Capítulo 5

O controlador de configurações RSCM

O objetivo principal deste trabalho é gerar uma infra-estrutura inicial para implementar SDRs. Tal estrutura é composta pelo controlador RSCM proposto dentre outros núcleos. No Capítulo anterior, tratou-se a estrutura do sistema proposto e o comportamento de cada um dos seus submódulos. No entanto, não foram tratados os detalhes de implementação do sistema, ou seja, quais estratégias foram adotadas com a finalidade de respeitar as características do sistema, definidas na proposta apresentada no Capítulo anterior. Assim sendo, nesse Capítulo, apresentam-se tais estratégias de implementação bem como a validação dos submódulos do controlador de configurações proposto.

Como visto anteriormente, os sistema RSCM é composto por seis sub-módulos:

- Monitor de Reconfiguração; (5.1)
- Escalonador de Configurações; (5.2)
- Autoconfigurador; (5.3)
- Interface de Configuração; (5.4)
- Controle Central de Configurações; (5.5)
- Memória de Configurações. (5.6)

Cada um desses submódulos tem sua implementação e validação detalhadas nas Seções 5.1 a 5.6 na ordem apresentada acima. Na Seção 5.7 apresenta-se a validação global do sistema RSCM. Por último, na Seção 5.8, descreve-se as ferramentas desenvolvidas durante esse trabalho.

Desde já, é importante salientar que todo sistema RSCM foi descrito em linguagem de descrição de hardware VHDL. Todos os fontes são disponibilizados em *http://www.inf.pucrs.br/~gaph.*

5.1 O Monitor de Reconfiguração

O Monitor de Reconfiguração (MR) monitora as fatias reconfiguráveis do sistema, verificando a existência de condições de reconfiguração. Essas condições são eventos que

habilitam a reconfiguração de bitstreams parciais do SDR, tal como o término da execução de uma dada configuração executando, seguida da conseqüente liberação de uma fatia. O comportamento de MR é representado pela máquina de transição de estados esboçada na Figura 5.1.

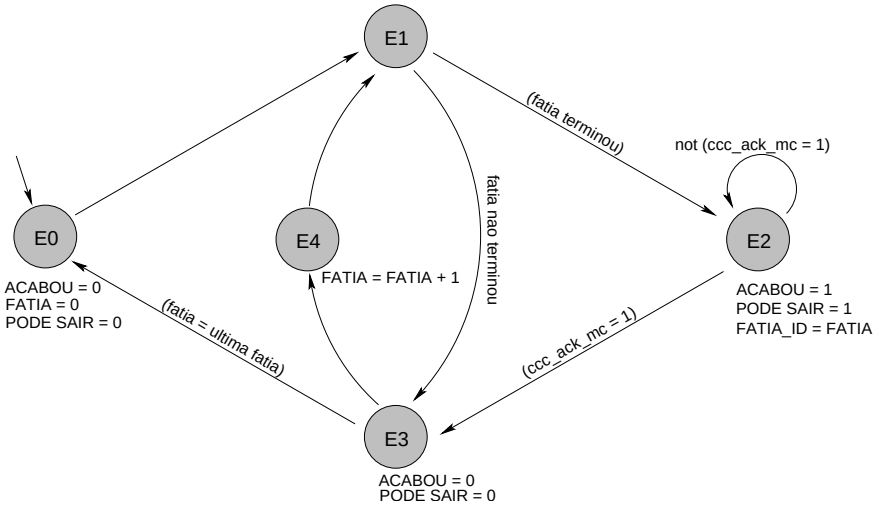

Figura 5.1: Máquina de estados do Monitor de Reconfiguração – *Cada uma das fatias reconfiguráveis do sistema é testada. A detecção de uma condição de reconfiguração leva MR a notificar o Controle Central de Configurações.*

A máquina de estados da Figura 5.1 compõe-se de cinco estados. No primeiro estado, **E0**, os sinais internos e de interface com fatias e o módulo CCC são inicializados. Entre esses sinais os principais são apresentados na Tabela 5.1. No estado **E1** é verificado se a primeira fatia está sinalizando que terminou sua execução. Se a fatia não estiver sinalizando *terminei*, MR passa ao estado **E3**. Caso contrário MR passa ao estado **E2**, onde envia o sinal de *acabou* e $fatia\_id$ para CCC. Nesse mesmo estado, MR envia o sinal de reconhecimento (i.e. $pode\_sair$) para fatia que acabou enquanto aguarda reconhecimento por parte do CCC (i.e. sinal ccc_ack_mr). Quando recebe o reconhecimento, MR passa ao estado **E3**. Nesse estado é verificado se existem outras fatias a serem testadas. Se existir alguma fatia, no estado **E4**, o índice de fatias é incrementado e MR retorna ao estado **E1**. Caso contrário MR passa ao estado **E0**, no qual o índice de fatia é reiniciado e todo o processo se repete.

MR testa as fatias reconfiguráveis uma a uma, verificando condições de reconfiguração. Sempre que uma condição é encontrada, após realizar as devidas operações, MR volta a testar as fatias, iniciando a partir da fatia seguinte aquela onde foi detectada uma condição. Dessa forma, todas as fatias serão testadas. Porém, a ordem de atendimento das condições pode não condizer com a ordem na qual as fatias sinalizam, ou seja, pode ocorrer da fatia $F1$ sinalizar seu término antes de $F5$ fazer o mesmo, mas a condição da última ser disparada antes, porque $F1$ terminou depois de ser testada, por exemplo. Assim, $F1$ só será atendida quando o sistema completar o ciclo de teste de todas as fatias e novamente

testar $F1$.

Tabela 5.1: Sinais manipulados no Monitor de Reconfiguração – *Os sinais $fatia\_id$ e acabou pertencem a interface entre MR e CCC. Os sinais terminei e $pode\_sair$ são da interface entre MR e as fatias reconfiguráveis do sistema. i é apenas um índice interno ao MR.*

Sinais	***Descrição***
$fatia\_id$	índice da fatia que terminou sua execução, enviado para o CCC
acabou	sinal utilizado para notificar o CCC que a configuração da $fatia\_id$ terminou sua execução
terminei	sinal recebido de um dada fatia como sinalização de que sua execução chegou ao final
$pode\_sair$	sinal enviado a uma dada fatia para notificá-la que sua sinalização de término (sinal *terminei*) foi detectada
i	índice da fatia a ser analisada

5.1.1 Validação do Monitor de Reconfiguração

Para validar a máquina de estados do Monitor de Reconfiguração foram realizadas simulações, para as quais foram desenvolvidos emuladores de comportamento de fatias reconfiguráveis e da interface entre CCC e MR. A forma de onda apresentada na Figura 5.2 ilustra parte da simulação realizada.

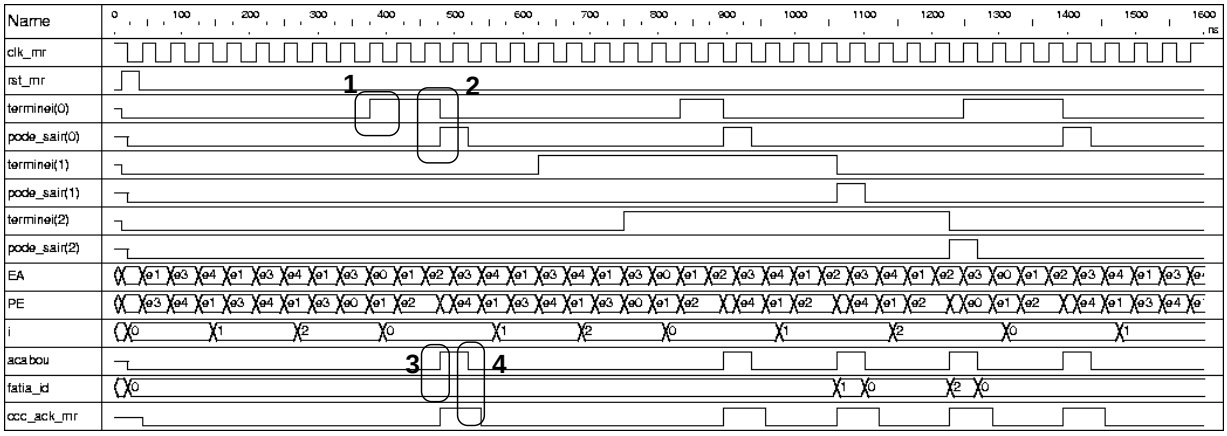

Figura 5.2: Forma de onda da simulação do módulo MR – **1** *é a ativação de terminei*(0). **2** *é a ativação de $pode\_sair(0)$ e a conseqüente desativação de terminei*(0). **3** *é a ativação de acabou e $fatia\_id$.* **4** *é a ativação de $ccc\_ac\_ + mr$ e conseqüente desativação de acabou*

A forma de onda da Figura 5.2 representa a operação de MR em um sistema reconfigurável composto por três fatias. Durante esta simulação deve-se perceber que em **1**, a configuração presente na fatia 0 notifica MR que terminou sua execução (i.e. $terminei(0) = 1$) e aguarda um sinal de reconhecimento dessa notificação. Em **2**, MR reconhece a condição

de reconfiguração disparada pela configuração da fatia 0 e imediatamente envia o sinal de reconhecimento (i.e. $pode\_sair(0) = 1$). MR, em **3**, notifica CCC que a configuração da fatia 0 (i.e. $fatia\_id(0) = 0$) acabou sua execução (i.e. $acabou = 1$) e aguarda o reconhecimento dessa notificação por parte de CCC. Esse reconhecimento (i.e. $ccc\_ack\_mr = 1$) é enviado por CCC em **4**. Todos esses passos refletem a detecção de uma condição de reconfiguração (i.e. término da configuração residente na fatia $F0$) e a notificação desse evento ao Controle Central de Configurações. Essas operações condizem com o comportamento desejado para MR.

Além de simular o módulo MR, também prototipou-se o sistema em FPGA. Durante essa fase, pôde-se perceber que o sistema portou-se adequadamente e apresentou resultados idênticos aos obtidos através da simulação. Em virtude disso, as formas de ondas relativas à prototipação não serão aqui apresentadas. O resultados foram analisados através da utilização de analisador lógico de formas de ondas HP1672G comercializado pela Hewlett-Packard Ltd.

5.2 O Escalonador de Configurações

O Escalonador de Configurações do sistema RSCM apresenta uma estrutura interna que mantém informações sobre a dependência entre as configurações do sistema. A operação do escalonador pressupõe a necessidade de percorrer esta estrutura para a seleção das configurações a serem escalonadas, de acordo com a requisição realizada pelo CCC. Essa estrutura é estática porque os dados manipulados não podem ser perdidos/sobrescritos, como no caso de sistemas com laços de execução. A única informação que sofre modificações durante a operação do sistema é o *número de predecessoras* de uma configuração. Quando uma configuração termina, o número de predecessores de cada uma das suas sucessoras é decrementado. Apenas esta informação pertencente a estrutura sofre modificações durante a operação do sistema, por isso esta é copiada para um vetor interno (i.e. VDI, Vetor de Descritores Internos), passível de atualizações.

Originalmente, RSCM aplica um escalonamento pré-definido, ou seja, emprega uma política de escalonamento estático. Não existe, no momento, qualquer possibilidade de escalonamento com preempção de configurações. O principal motivo para não empregar técnicas de preempção é não encontrar justificativas suficientes para implementar sistemas que necessitem dessa estratégia complexa. SDRs e o controlador de configurações têm suas complexidades aumentadas com o emprego de técnicas de preempção devido à necessidade de prover estruturas e métodos de salvamento e controle de contexto de configurações.

A Figura 5.3 apresenta a lógica do Escalonador de Configurações como uma máquina de estados. No estado inicial **E0**, as variáveis internas e sinais enviados ao CCC são inicializados. Entre esses sinais, os principais estão listados e definidos na Tabela 5.2.

Tabela 5.2: Sinais manipulados no Escalonador de Configurações – *Os três primeiros sinais pertencem a interface entre EC e o Controle Central de Configurações, enquanto os demais são sinais internos utilizados para armazenamento temporário de informações*

Sinais	***Descrição***
$ocupado\_ec$	sinal enviado ao CCC que reflete o estado de ocupação do Escalonador
ec_ack_ccc	sinal de reconhecimento da requisição por escalonamento enviada por CCC
$prox\_conf\_id$	identificador da configuração escalonada por EC
$configurar$	sinal que notifica CCC que a configuração $prox\_conf\_id$ pode ser configurada
cid	índice temporário de configuração, utilizado durante a inicialização do VDI
sid	índice temporário de sucessores, utilizado durante a procura da configuração a ser escalonada
$acfg$	sinal temporário que armazena o identificador da configuração que terminou sua execução, recebe ant_conf_id, enviado por CCC durante a requisição dos serviços de EC

Os estados **E1**, **E2** e **E3** inicializam o VDI (i.e. Vetor de Descritores Internos) com informações sobre o número de predecessores de cada uma das configurações do sistema. Uma vez que todas as posições do vetor foram inicializadas, EC passa ao estado **E4**, onde aguarda CCC requisitar seus serviços através do sinal req_ec. Após receber a requisição, EC passa ao estado **E5**, onde $ocupado\_ec$ é ativado, o código da configuração terminada é armazenado e seu número de predecessores é restaurado para seu valor original. Essa restauração possibilita a realização de laços de configurações. No estado **E6**, o escalonador envia a CCC o reconhecimento da requisição de seus serviços. A partir do estado **E7** a tarefa de escalonamento é realmente realizada. Nesse estado, o identificador do primeiro sucessor é copiado a partir da tabela de dependência e descritores (TDD). Em **E8**, o número de predecessores desse sucessor é decrementado e, testado em **E9**. Se for zero EC passa ao estado $E12$ e envia o código do sucessor encontrado para CCC junto ao sinal $configurar$, sinalizando que esse sucessor deve ser configurado. EC continua nesse mesmo estado até que CCC reconheça a notificação, através do sinal ccc_ack_ec. Após receber o reconhecimento ou se o sucessor não estiver apto a ser escalonado, EC passa ao estado **E10**, que é responsável por verificar se a configuração terminada possui outros sucessores. Se não existirem sucessores, EC retorna ao estado **E4**. Caso contrário sid é incrementado e o processo é novamente realizado, a partir do estado **E7**.

5.2.1 Validação do Escalonador de Configurações

O escalonador de configurações foi validado funcionalmente e também prototipado. Diversos cronogramas de escalonamentos foram testados e a forma de onda apresentada na Figura 5.4 ilustra a operação do Escalonador de acordo com o cronograma apresentado na Figura 4.3.

Em **1**, VDI é inicializado com o número de predecessores de cada uma das configurações. Em **2**, CCC notifica que a configuração $C0$ acabou de executar e requisita os serviços de EC. O escalonador, por sua vez, reconhece a requisição em **3**. No evento **4**, EC encontra o

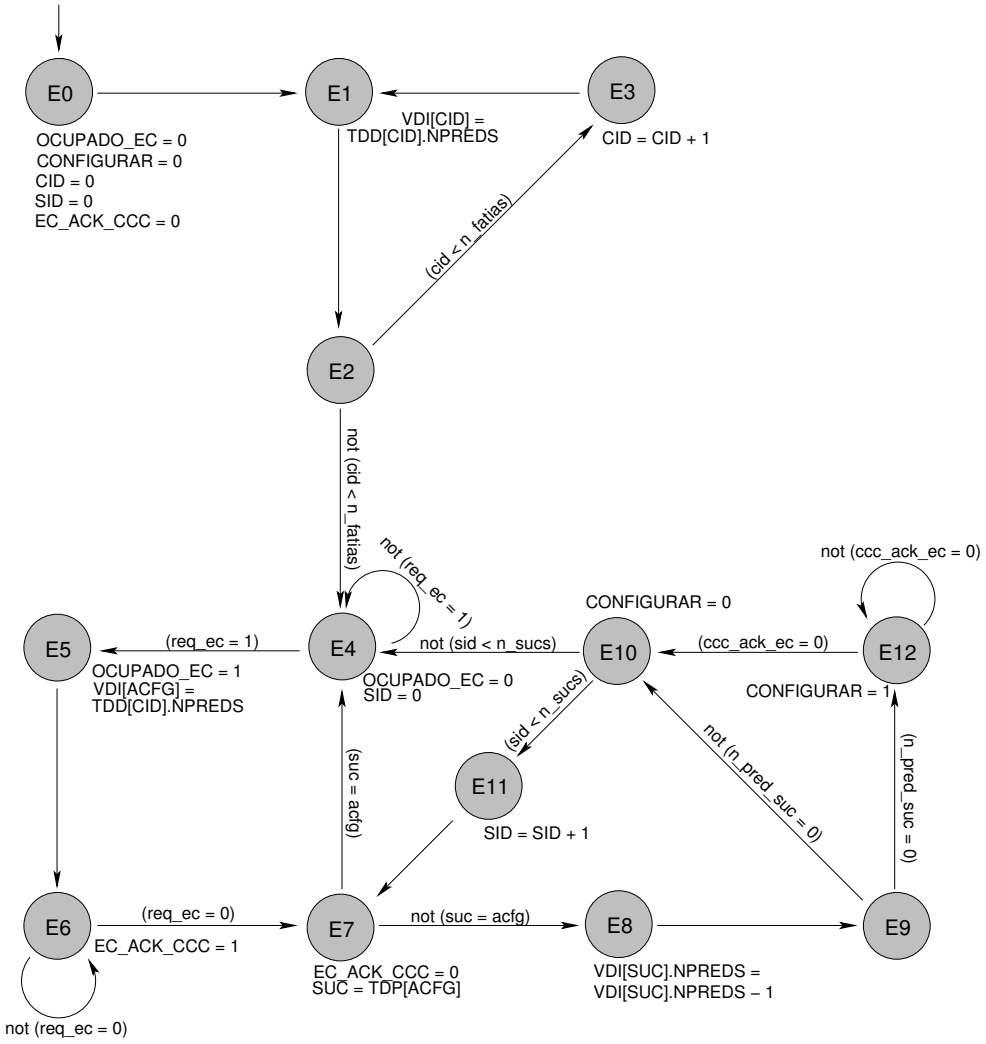

Figura 5.3: Máquina de estados do Escalonador de Configurações – *Esta máquina aguarda requisições de escalonamento. Logo após, atualiza as informações de número de predecessores do VDI e testa se configurações devem ser escalonadas. EC escalona as configurações cujo número de predecessores é zero, ou seja, aquelas com todos predecessores terminados.*

primeiro sucessor da configuração que terminou de executar, nesse caso a configuração $C1$. Seguindo, o número de predecessores de $C1$ é decrementado. Como o valor tornou-se zero, vide **5**, EC solicita à CCC a configuração desse sucessor, como apresentado em **6**. No evento **7**, CCC reconhece a requisição de EC. Seguindo, a próxima sucessora da configuração $C0$ também é escalonada. Pode-se perceber que o escalonamento aplicado está de acordo com a grafo apresentado na Figura 4.3 (página 62), ou seja, quando a configuração $C0$ termina, as configurações $C1$ e $C2$ são escalonadas. Quando a $C1$ termina, $C5$ é escalonada. Quando $C2$ termina, $C5$ é escalonada. Quando $C5$ termina, nada acontece porque essa configuração não possui sucessores. Nada acontece também quando $C3$ termina, porque seu sucessor,

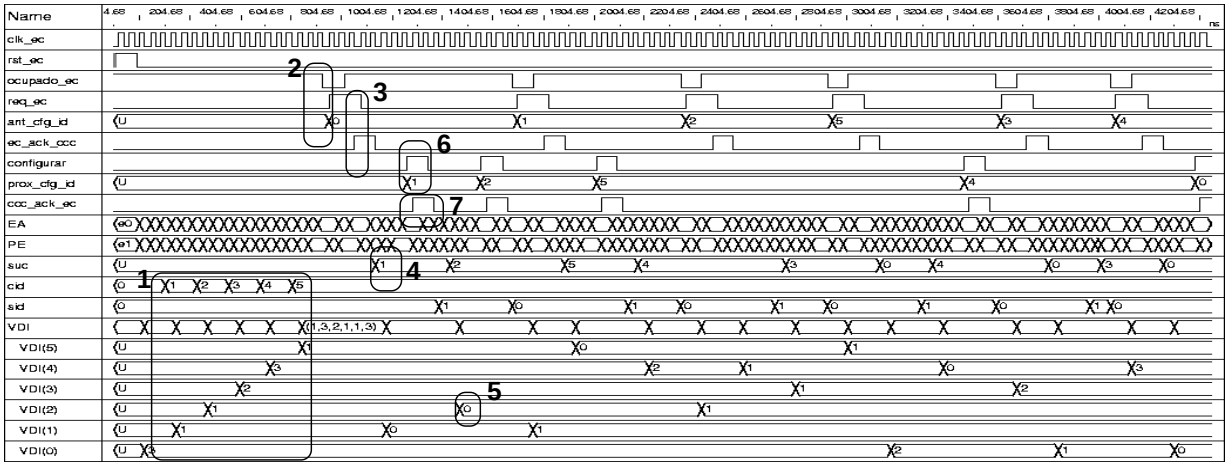

Figura 5.4: Forma de onda da simulação do módulo EC – **1** *é a inicialização de VDI.* **2** *é a requisição par EC.* **3** *é o reconhecimento da requisição.* **4** *indica que o sucessor* $C1$ *deve ser testado.* **5** *mostra que o número de predecessores de* $C1$ *é zero.* **6** *é a requisição da reconfiguração de* $C1$*.* **7** *é o reconhecimento desta requisição. Acompanhando a forma de onda pode-se perceber que EC comportou-se de acordo com o cronograma apresentado na Figura 4.3.*

$C0$, não depende apenas dela e só é escalonado quando $C4$ termina.

Os mesmos casos simulados foram prototipados em FPGA e apresentaram os mesmos resultados da simulação. Assim, pôde-se certificar o correto funcionamento do Escalonador de Configurações do sistema RSCM.

5.3 O Autoconfigurador

A implementação do Autoconfigurador implicou o desenvolvimento de duas máquinas de estados e da estrutura que realiza a interface entre elas. A primeira máquina recebe requisições de reconfiguração de CCC, captura os dados da Memória de Configurações e os armazena em um buffer interno, denominado Buffer De Palavras (BDP). A outra máquina lê os dados desse buffer e os envia à Interface de Configuração.

Propõe-se aqui o desenvolvimento de um sub-módulo responsável por aplicar relocação de configurações através da modificação do arquivo de configuração parcial em tempo de execução. Esse módulo eliminaria a necessidade de armazenar diversas instâncias de cada uma das configurações, uma para cada fatia reconfigurável do dispositivo. Devido às restrições de tempo impostas pelo cronograma do trabalho não foi possível implementar tal módulo até o momento. Além disso, a carência de documentação sobre a estrutura de configuração de dispositivos da família Virtex-II dificultou demasiadamente o desenvolvimento dessa tarefa. No entanto, a relocação de configurações pode ser simulada armazenando diversas instâncias de um bitstream parcial. O desenvolvimento do módulo Relocador constitui uma otimização do sistema, evidenciada desde já como trabalho futuro.

5.3.1 Organização dos dados na memória

Para facilitar a leitura de dados de configuração da memória, uma estrutura de armazenamento dos arquivos parciais foi definida. A memória é subdividida em duas partes: ponteiros e dados de configurações. A área de *ponteiros*, como o nome indica, possui ponteiros para o início de cada um dos arquivos de configuração. Por exemplo, o endereço 2 é um ponteiro para o início dos dados da configuração $C2$. O tamanho da área de ponteiros é determinado pelo número de configurações parciais constituintes do SDR, incluindo todas as instâncias de cada configuração. A área de *dados de configuração* contém um ponteiro para o endereço final da configuração seguido dos dados do arquivo a ser lido. Na Figura 5.5 apresenta-se um resumo da organização de dados na memória definida.

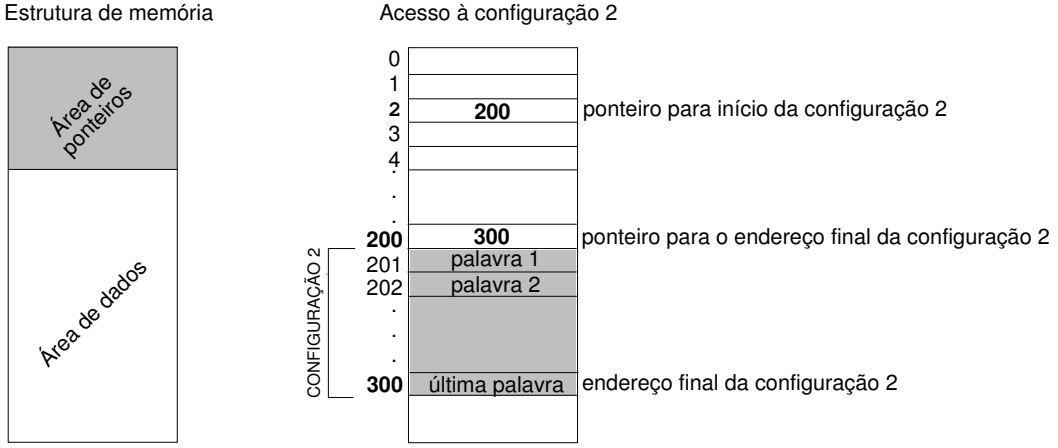

Figura 5.5: Organização dos dados na Memória de configurações – *Para acessar a configuração $C2$ na memória. AC deve ler a posição 2 de memória, que contém um apontador para o início dos dados de configurações. A primeira palavra indica a posição final a ser lida durante a reconfiguração.*

Como é necessário armazenar uma instância de cada configuração para cada uma da fatias reconfiguráveis devido à ausência de relocabilidade de bitstreams, foi necessário definir uma política para acessar o arquivo de configuração correto na memória. Cada configuração é acessada não apenas por seu identificador, mas também é aplicado um deslocamento de acordo com a fatia alvo. A Equação 5.1 é utilizada para acessar o arquivo de configuração desejado.

$$End = id_{conf} + (id_{fat} \times N_{confs}) \tag{5.1}$$

O Endereço (End) é composto pela soma entre o identificador da configuração escalonada (id_{conf}) e o produto entre a identificação da fatia alvo (id_{fat}) e o número de configurações do sistema (N_{confs}).

5.3.2 O Leitor de Configurações do AC

O comportamento do módulo que lê dados de configurações da Memória de configurações e os armazena no buffer é apresentado na Figura 5.6. Nessa máquina, os estados **M0** e **M1** aguardam a requisição de reconfiguração enviada por CCC e inicializam sinais, dentre eles os listados na Tabela 5.3.

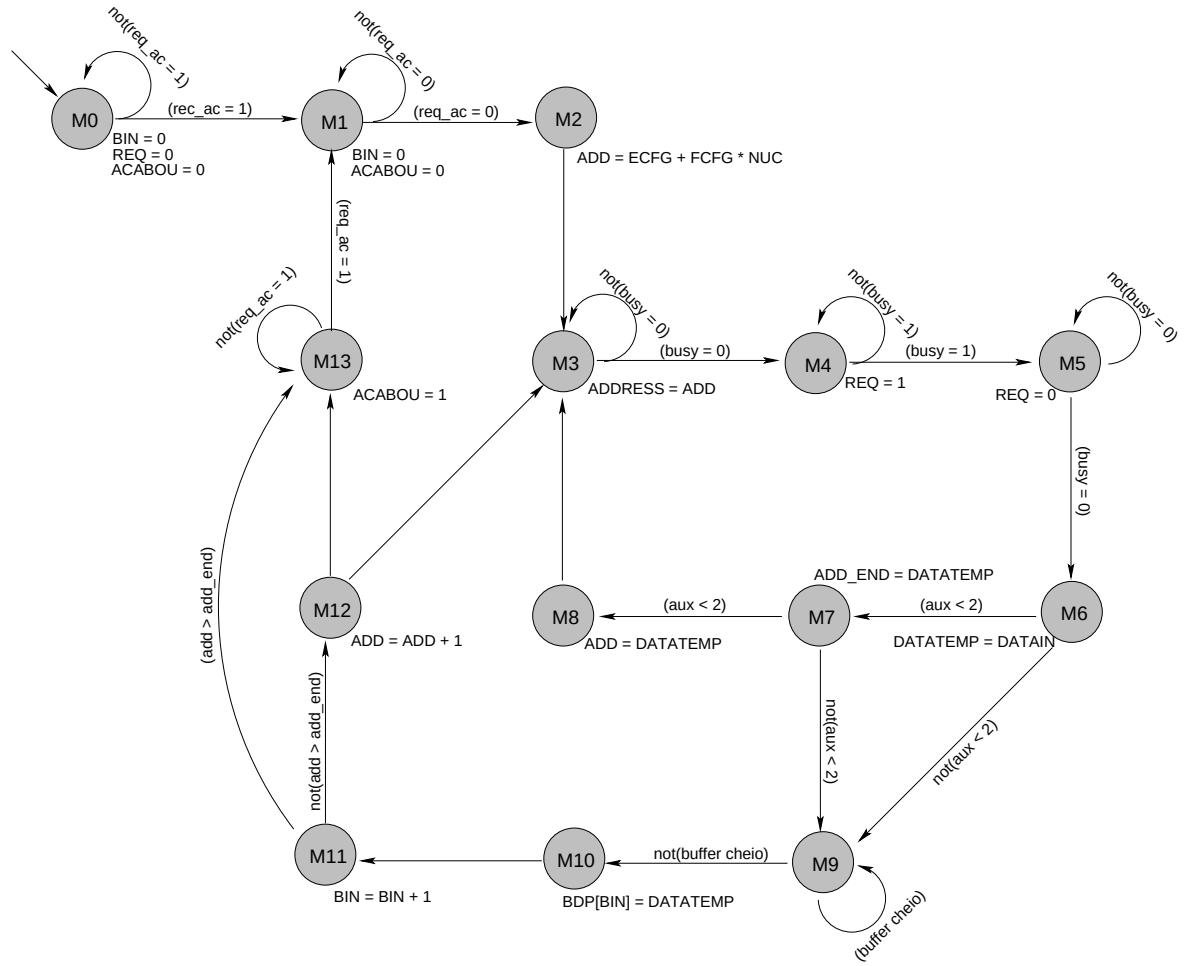

Figura 5.6: Máquina de estados do Leitor de Configurações do AC – *Esta máquina calcula o endereço do bitstream parcial solicitado e lê da MC os dados, armazenando-os em BDP.*

Em **M2**, calcula-se a posição do apontador para a posição inicial do arquivo de configuração. No estado **M3** *add* é enviado para os pinos de endereço da memória e verifica-se a ocupação da memória. A requisição de serviços da memória é realizada no estado **M4**. Aguarda-se a operação de leitura ser finalizada em **M5**. No estado **M6** o dado lido da memória é internamente armazenado e o teste que verifica se o dado lido é um ponteiro é realizado. No estado **M7** AC inicializa o endereço final do arquivo de configuração (i.e *add_end*). No estado **M8** AC encarrega-se de inicializar o endereço inicial. Em **M9**, verifica-se a existência de posição livre no buffer. Em **M10**, os dados lidos a partir da

Tabela 5.3: Sinais manipulados no Leitor de Configurações do AC – *Os três primeiros sinais pertencem a interface para a Memória de Configurações. Os demais são utilizados para acesso ao buffer e sinalização entre as duas máquinas de AC.*

Sinais	***Descrição***
req	sinal usado para requisitar serviços da memória
we	sinal que seleciona operação realizada pela memória (leitura/escrita)
add	endereço a ser acessado na memória, inicializado com o próprio identificador da configuração requisitada, de acordo com a organização de dados apresentada na Figura 5.5
bin	ponteiro para posição de escrita no buffer de palavras (i.e. BDP)
acabou	sinal utilizado para notificar a máquina de Configuração de AC que todos os dados do arquivo foram escritos no buffer

memória são armazenados no buffer de palavras. Em **M11**, verifica-se o término do arquivo de configuração e incrementa-se o ponteiro para posição de escrita no buffer (i.e. *bin*). Caso o arquivo não tenha terminado, no estado **M12** o endereço de leitura é incrementado. Caso contrário, no estado **M13**, *acabou* é setado para notificar a outra máquina do AC, o Configurador, que todos dados estão no buffer. Em seguida, a leitura da memória acaba.

5.3.3 O Configurador do AC

A máquina de estados que lê os dados de configuração do buffer e os envia à Interface de Configuração possui a estrutura descrita na Figura 5.7. No estado **R0** AC aguarda a requisição de reconfiguração proveniente do CCC e inicializa os sinais da Interface de Configuração, entre eles os apresentados na Tabela 5.4.

Tabela 5.4: Sinais manipulados no Configurador do AC – *Os dois primeiros sinais pertence a interface com CCC. Os quatro seguinte dizem respeito a interface com a IC. O último armazena um valor temporário.*

Sinais	***Descrição***
ocupado_ac	sinal enviado para CCC que expressa o estado do AC
ac_ack_ccc	sinal transmitido a CCC, reconhecendo a requisição de reconfiguração realizada por CCC
habilitar_ic	sinal que habilita a operação da IC
operar_ic	sinal que dita a freqüência de operação da IC
write_ic	sinal que seleciona a configuração/readback do dispositivos através da IC
entrada_ic	barramento de dados enviados para a IC (8 bits)
bout	ponteiro para posição de leitura no BDP

No estado **R1** o sinal de reconhecimento (i.e. *ac_ack_ccc*) é enviado à CCC. Em **R2**, verifica-se a existência de dado a ser lido do buffer. Quando existir dado, AC passa ao estado **R3**, onde aguarda a Interface de Configuração não estar ocupada (i.e. *ocupado_ic*). A seguir, um byte da palavra de dados é selecionado para ser enviado à IC, de acordo com

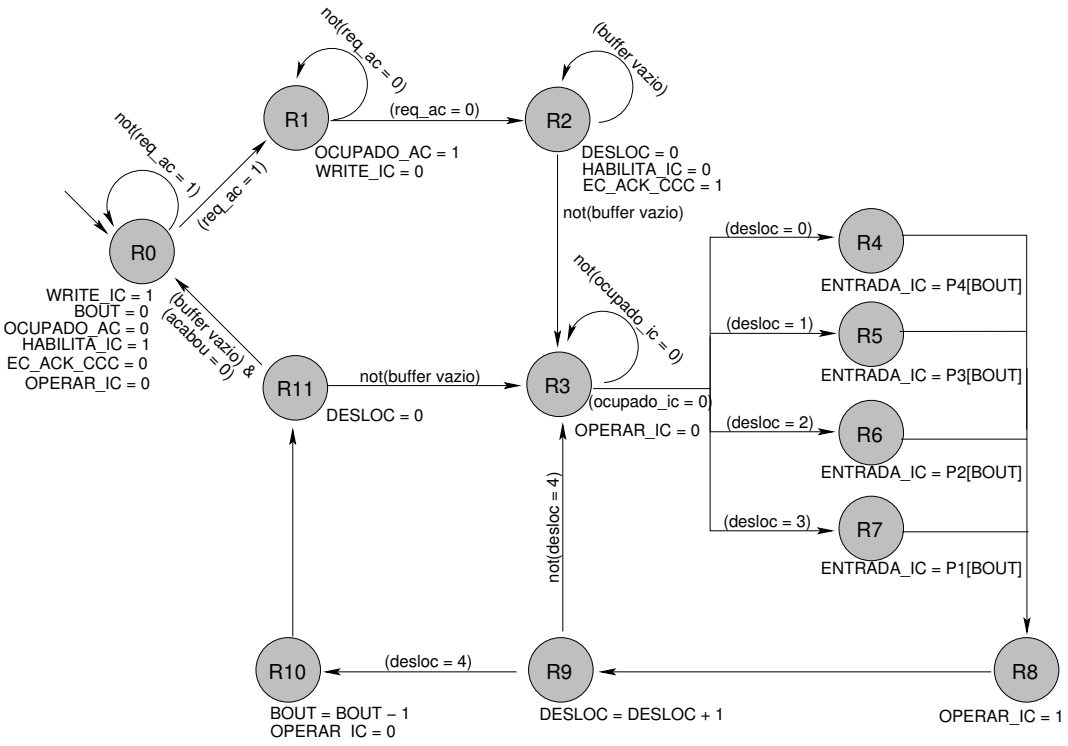

Figura 5.7: Máquina de estados do Configurador do AC – *Esta máquina lê os dados de configuração armazenados pela máquina da Figura 5.6 e envia-os para a IC, de maneira a reconfigurar parcialmente o dispositivo com o bitstream parcial solicitado.*

o sinal interno *desloc*. Os estados **R4**, **R5**, **R6** e **R7** servem para enviar cada um dos 4 bytes da palavra de dados para a IC. Em **R8**, o sinal *operar_ic* é ativado, para notificar IC que um dado deve ser enviado à máquina de configuração do dispositivo. O sinal *desloc* é testado a fim de verificar se todos os bytes da palavra foram enviados. Se isso não aconteceu AC retorna ao estado **R3**. Caso contrário, o AC passa ao estado **R10**, onde *bout* é incrementado. Em seguida, AC passa ao estado **R11**, onde novamente verifica-se a existência de dado a ser lido em BDP. Se houver, AC retorna ao estado **R3**. Caso contrário, testa-se *acabou* para certificar se todos os dados estavam escritos no buffer. Se *acabou* estiver ativo, o processo de configuração é encerrado e o sinal *ocupado_ac* é desativado.

5.3.4 Validação do Autoconfigurador

Assim como os módulos anteriormente apresentados, o Autoconfigurador também foi validado e prototipado em FPGA. As simulações comportaram-se da forma esperada, mas durante a prototipação diversos problemas foram encontrados. Em virtude disso, foram analisados todos os sinais da interface SelectMAP durante um processo de configuração utilizando o software Impact do ambiente ISE da Xilinx. A partir daí, desenvolveu-se um hardware emulador de SelectMAP. Esse módulo portou-se de forma correta de acordo

com os testes realizados. Um importante avanço foi obtido quando alguns eventos foram percebidos e posteriores melhorias realizadas no emulador, dentre estes:

- Identificou-se que o tempo total de configuração poderia ser reduzido a 1/3 do tempo original obtido na configuração via software. Isso foi possível porque os primeiros quatro pulsos de relógio de configuração possuíam uma freqüência desnecessariamente baixa em relação aos demais que compunham o processo de configuração em si. Este fato é demonstrado na Figura 5.8 em **1**. O desempenho do sistema melhora se a freqüência for sempre a mais alta para todos os dados enviados;
- Identificou-se também que os pulsos de relógio após o último pulso que envia dados de configuração eram desnecessários, de acordo com **2** da Figura 5.8. Dessa forma, pode-se remover estes ciclos;
- Gradativamente aumentou-se a freqüência de operação do emulador até 12 MHz.

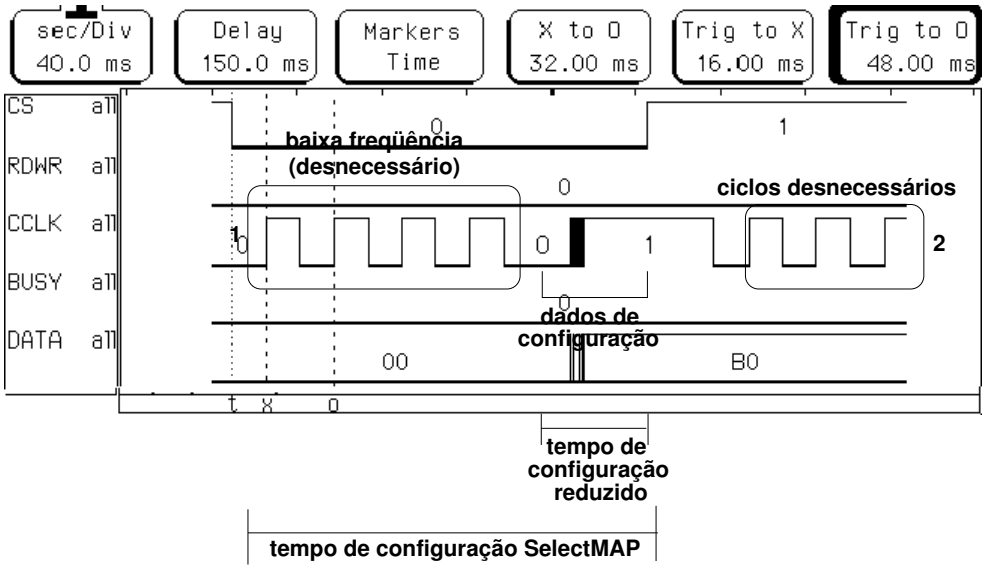

Figura 5.8: Interface de configuração SelectMAP capturada. – *Em 1 tem-se a primeira palavra enviada para a interface de configuração do dispositivo, a uma freqüência desnecessariamente baixa. Em 2 existem pulsos de configuração desnecessários.*

Para certificar a correto funcionamento desse emulador foi implementado um sistema em duas placas de prototipação. A primeira delas contém a lógica reconfigurável. A segunda contém o emulador SelectMAP. As duas plataformas foram conectadas, ligando os dados gerados pelo emulador à interface SelectMAP da outra plataforma através de fios externos. Assim, a segunda plataforma reconfigura parcialmente o dispositivo da primeira plataforma, transformando a lógica original desta.

Após adaptar o Autoconfigurador para se comportar de maneira análoga ao emulador, pôde-se obter o funcionamento correto do AC. Este é ilustrado na Figura 5.9. As máquinas

de estados do Leitor e do Configurador do módulo AC são monitoradas nessa simulação, onde foram utilizados arquivos de configuração hipotéticos, com apenas oito palavras de dados. Antes dos dados de configuração propriamente ditos, é necessário enviar à IC três palavras:

- uma palavra qualquer, nesse caso AC está enviando sempre o endereço final da configuração;
- uma palavra de preenchimento, sempre o valor hexadecimal **FFFFFFFF**;
- uma palavra de sincronização da configuração, o valor hexadecimal **AA995566**.

A máquina de configuração dos dispositivos VirtexII ignora quaisquer outras palavras enviadas antes dessas três palavras.

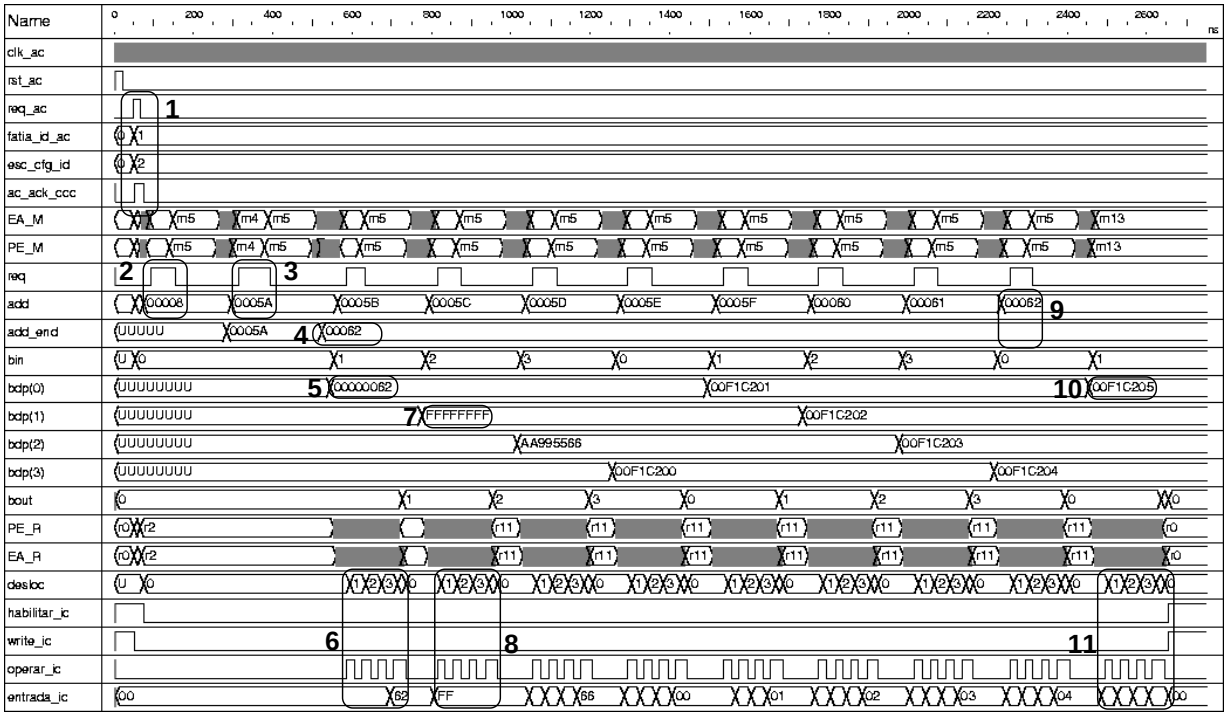

Figura 5.9: Forma de onda da simulação do módulo AC – *A requisição de reconfiguração é enviado a AC em* **1**, *que calcula o endereço da configuração na MC em* **2** *e* **2**. *AC lê dados de configuração de MC e envia-os para IC durante* **4** *a* **6**. *Este processo se repete para as próximas 8 palavras de dados, terminando o processo em* **11**.

Nessa simulação, os seguintes onze eventos podem ser observados. No evento **1**, CCC requisita ao AC a reconfiguração da configuração $C2$ na fatia $F1$. Imediatamente após, AC reconhece essa requisição. Em **2**, é calculado o endereço correspondente ao apontador para o início da configuração e fatia desejadas. Logo após, esta posição de memória é lida. O endereço inicial da leitura (*add*) é inicializado em **3**, enquanto que o endereço final (*add_end*) é inicializado em **4**. No evento **5**, a primeira palavra de configuração é escrita no buffer de palavras (BDP). A seguir, em **6**, cada um dos bytes de configuração é enviado

à Interface de Configuração. Em **7** e **8**, a segunda palavra é escrita em BDP e enviada à IC. Esse processo se repete até que seja verificado que o endereço atual é igual ao endereço final, em **9**. Então, a última palavra é escrita no buffer em **10**. Em seguida, a mesma é enviada para a IC, em **11**, quando termina o processo de reconfiguração.

Estes eventos certificam o correto funcionamento de AC. O mesmo comportamento obtido na simulação apresentada na Figura 5.9 foi observado na prototipação em FPGA do AC.

5.4 A Interface de Configuração

O módulo Interface de Configuração possui uma instância do componente ICAP (*Internal Configuration Access Port*) [XIL02c] para que o usuário acesse internamente a porta de configuração do dispositivo. Esse módulo é documentado de forma muito precária pelo fabricante de FPGAs Xilinx. As poucas informações sobre ele definem sua interface como sendo igual a do modo de configuração paralelo dependente, particularmente denominado pela Xilinx de slave SelectMAP. Tal interface é composta pelos sinais apresentados na Tabela 5.5.

Tabela 5.5: Conjunto de sinais que compõem a interface ICAP – *Esta interface permite acessar internamente a porta de configuração de disitivos da família VirtexII e VirtexII-Pro da Xilinx. Segundo a mesma, clk pode possuir um frequencia máxima de 33MHZ.*

Sinal	***Sentido***	***Largura* (bits)**	***Descrição***
clk	entrada	1	freqüência de configuração ou readback
ce	entrada	1	habilita operação da ICAP
write	entrada	1	seleciona tipo de operação, (ZERO = configuração)
i	entrada	8	dados de configuração
o	saída	8	dados lidos do dispositivo (readback)
busy	saída	1	status de ocupação da máquina de configuração

A freqüência máxima de configuração utilizando ICAP é 33MHz. Infelizmente, até o momento não foi possível utilizar o componente ICAP no sistema. Para contornar o problema, foi desenvolvido um emulador de SelectMAP, conforme a Subseção 5.3.4. Trata-se de um módulo que configura o dispositivo através da interface externa de configuração, utilizando fios como ponte, como apresentado anteriormente. Esse mesmo sistema foi adaptado para enviar os dados para a ICAP e a configuração não funcionou corretamente. Ainda estão sendo realizados estudos para investigar se a ICAP realmente possui a mesma interface SelectMAP como informado pelo fabricante ou se algum detalhe não foi mencionado pelo mesmo. A importância da interface interna para a configuração diz respeito à redução do tempo de reconfiguração, e para reduzir a contagem de componentes do sistema. Estes são fatores determinantes para a avaliação da validade de desenvolver SDRs.

Como verificado em experimentos realizados com configurações paralelas (SelectMAP)

a palavra de dados é enviada byte a byte. A ordem de envio dos bits de cada byte é invertida, ou seja, no caso da palavra 01234567 a ordem de envios dos bytes é 01-23-45-67 mas o bit mais significativo de cada byte corresponde ao bit menos significativo da entrada SelectMAP (i.e. entrada não recebe o valor hexadecimal 01, mas sim o valor hexadecimal 80).

Sendo a Interface de Configuração um módulo simples que instancia um outro módulo validado pelo fabricante, não serão aqui apresentadas formas de ondas de simulação e prototipação desse módulo. A simulação portou-se adequadamente e a prototipação apresentou os problemas citados.

5.5 O Controle Central de Configurações

O Controle Central de Configurações coordena a operação dos demais módulos do RSCM. Recebe notificações do Monitor de Reconfiguração, solicita escalonamento de configurações ao Escalonador, e solicita reconfigurações parciais ao Autoconfigurador. Como mencionado anteriormente, para melhor coordenar a operação dos demais módulos do sistema, CCC foi subdividido em três submódulos.

A *Lógica de Controle* é composta por uma estrutura que armazena o estado de alocação do sistema, denominada Tabela de Alocação de Recursos (TAR) e um buffer que armazena as configurações escalonadas, o Buffer de Configurações Escalonadas (BCE). Os demais módulos, *Módulo Escalonamento* e *Módulo Configuração* são duas máquinas de estado, tratadas nas próximas Seções.

5.5.1 O Módulo Escalonamento do CCC

O *Módulo Escalonamento* realiza a interface de CCC com MR e EC. Esse módulo recebe notificações de MR, solicita escalonamento e armazena identificadores de configurações a serem escalonadas em BCE. A máquina de transição de estados da Figura 5.10 expressa o comportamento do Módulo Escalonamento do CCC.

Nessa máquina, diversos sinais e as interfaces com MR e EC são gerenciados, dentre eles: $fatia\_id$, *acabou* e $pode\_sair$ apresentados na Tabela 5.1, $ocupado\_ec$, ec_ack_ccc, $prox\_conf\_id$ e $configurar$ apresentados na Tabela 5.2, e outros, apresentados na Tabela 5.6.

No estado **E0**, alguns sinais são inicializados enquanto é verificado se EC está ocupado. Quando EC não está ocupado, CCC passa ao estado **E1**, onde espera uma notificação de *acabou* proveniente de MR. Após receber a notificação, CCC envia o sinal de reconhecimento para MR e armazena internamente o identificador da fatia liberada, no estado **E2**. Em **E3**, CCC requisita os serviços de EC (i.e. req_ec), envia o identificador da configu-

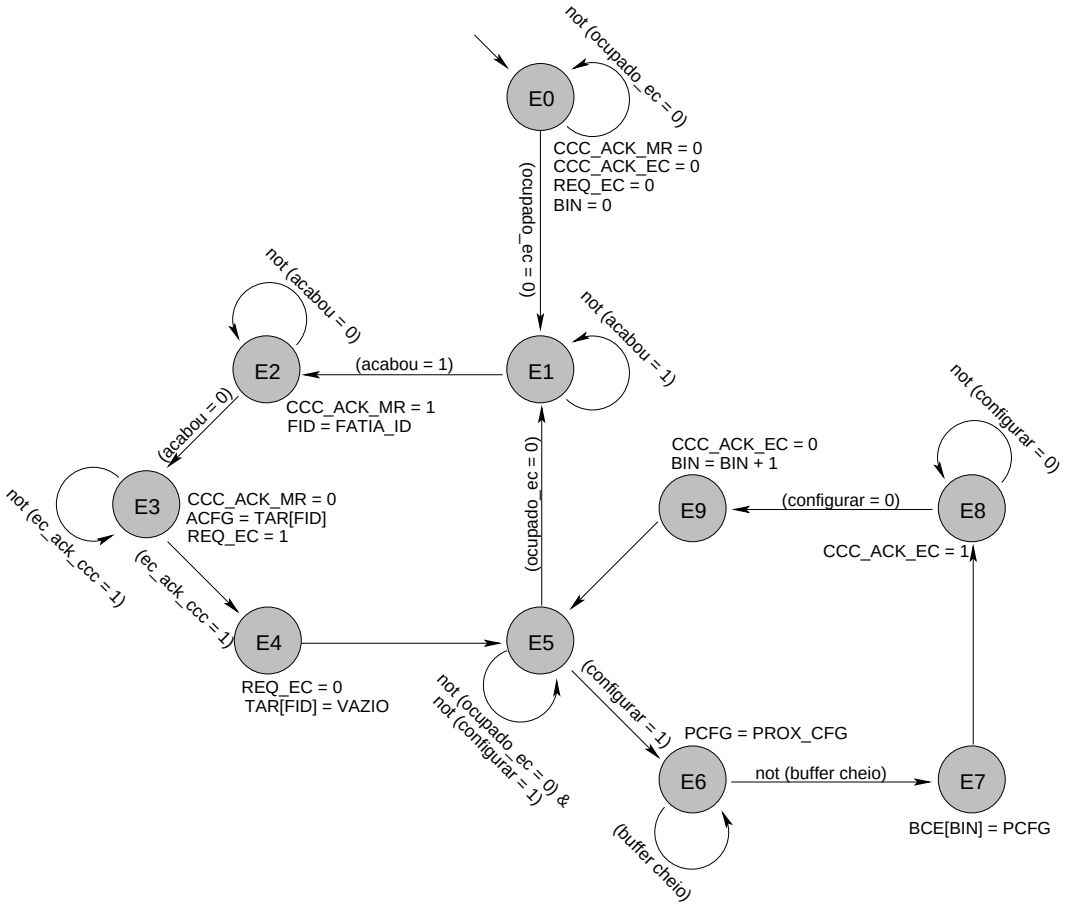

Figura 5.10: Máquina de estados do Módulo Escalonamento do CCC – *CCC aguarda notificações provenientes do MR. A seguir, requisita o escalonamento de configurações e aguarda a operação do EC. O resultado da operação do EC, é interpretado por CCC, que armazena em BCE as configurações escalonadas.*

Tabela 5.6: Sinais manipulados no módulo de Escalonamento do CCC – *Estes, sinais pertencem ao conjunto de sinais internos ao CCC, utilizados para armazenamento temporário. As interfaces com os outros módulos do RSCM, já foram tratadas anteriormente, nas Tabelas 5.1 e 5.2.*

Sinais	***Descrição***
bin	ponteiro para posição de escrita no buffer de configurações escalonadas
fid	sinal utilizado para armazenar internamente o identificador da fatia que terminou sua execução
pcfg	sinal utilizado para armazenar internamente o identificador da configuração escalonada por EC
acfg	sinal utilizado para armazenar internamente o identificador da configuração que terminou sua execução
vazio	constante interna que possui um valor que, quando escrito em uma posição da TAR, representa que não existe configuração alguma naquela fatia.

ração terminada para EC e aguarda o reconhecimento por parte do mesmo. Após receber o reconhecimento (i.e. *ec_ack_ccc*), CCC passa ao estado **E4**, onde TAR é atualizada, inserindo *vazio* na posição da fatia que terminou. Em **E5**, CCC aguarda a resposta de EC, a qual pode se dar de duas formas: na primeira, nenhuma configuração é escalonada, EC é desocupado (i.e *ocupado_ec* desativado) e CCC retorna ao estado **E1**. A segunda forma é EC escalonar uma configuração, quando CCC passa ao estado **E6**. Nesse estado, verifica-se a existência de posição livre para escrever em BCE. Se existir, o identificador da configuração escalonada é copiado. A seguir, no estado **E7**, esse identificador é escrito em BCE. O reconhecimento da operação de EC é enviado no estado **E8**. O ponteiro para escrita em BCE é incrementado no estado **E9**. Então, o CCC retorna ao estado **E5**.

5.5.2 O Módulo Configuração do CCC

O *Módulo Configuração* é a máquina de estados de CCC que lê identificadores de configurações escalonadas armazenados em BCE, procura fatias livres e requisita os serviços do AC. Seu comportamento corresponde à máquina de transição de estados apresentada na Figura 5.11.

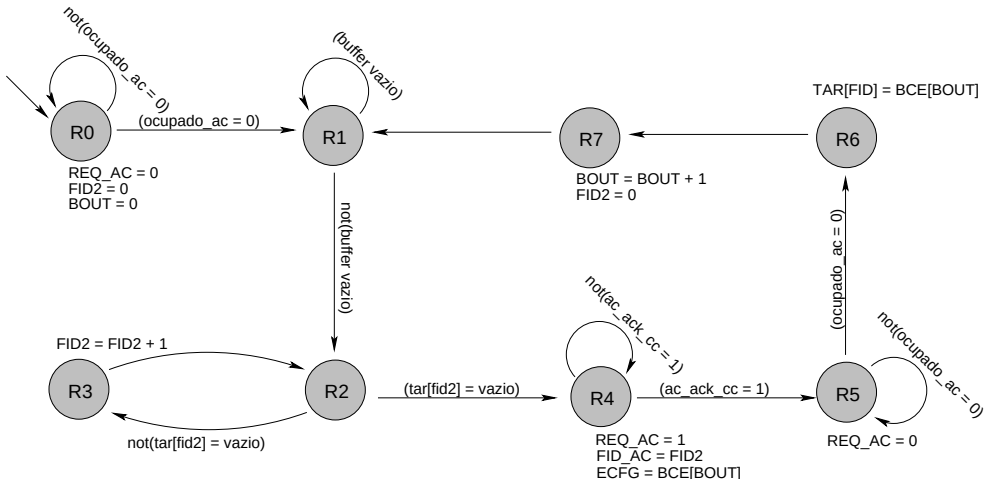

Figura 5.11: Máquina de estados do Módulo Configuração do CCC – *Quando existe configuração não configurada no BCE, CCC requisita sua configuração ao AC e aguarda o término da reconfiguração.*

Nessa máquina, diversos sinais e interfaces com a MC são gerenciadas, dentre eles os apresentados na Tabela 5.7. Os demais sinais manipulados (i.e. *ocupado_ac* e *ac_ack_ccc*) já foram apresentados na Tabela 5.3.

No estado inicial, **R0**, sinais são inicializados e a ocupação do AC é verificada. Quando AC não estiver ocupado, CCC passa ao estado **R1**, onde verifica-se a existência de configurações em BCE. Se existir alguma configuração em BCE, CCC segue ao estado **R2** e alterna-se em laços com o estado **R3**, procurando uma fatia livre. Quando encontra, copia seu identificador e sai do laço, passando ao estado **R4**. Nesse estado, CCC requisita serviços do AC e aguarda seu reconhecimento. Após receber esse reconhecimento (i.e.

Tabela 5.7: Sinais manipulados no módulo de Configuração do CCC – *Estes, sinais pertencem ao conjunto de sinais internos ao CCC, utilizados para armazenamento temporário. As interfaces com os outros módulos do RSCM, já foram tratadas anteriormente, na Tabela 5.3.*

Sinais	***Descrição***
$fid2$	índice de fatia utilizado para procurar fatias vazias, cujo valor é enviado ao AC, juntamente com a requisição por reconfiguração req_ac.
$bout$	apontador para a posição de leitura do buffer de configurações escalonadas (BCE)
$ecfg$	identificador interno da configuração escalonada, lida de BCE e enviada para AC

ec_ack_ccc), passa ao estado **R5**, onde aguarda que o AC termine a tarefa de reconfiguração. Quando essa termina, no estado **R6**, CCC atualiza TAR e passa ao estado **R7**. Nesse estado, CCC incrementa o ponteiro de leitura de BCE. Finalmente, a máquina retorna ao estado **R1**.

5.5.3 Validação do Controle Central de Configurações

O Controle Central de Configurações foi validado e prototipado em FPGA. Na Figura 5.12, apresenta-se a forma de onda que expressa o comportamento do Controle Central de Configurações obtido tanto durante a simulação funcional quanto durante a prototipação. Acompanhando o conteúdo do sinal TAR, pode-se monitorar o gerenciamento da ocupação das fatias do sistema. Neste caso, foram utilizadas duas fatias reconfiguráveis. Da mesma forma, acompanhando o sinal BCE, pode-se observar o gerenciamento da lista de configurações escalonadas pelo EC.

Nessa forma de onda é importante ressaltar sete eventos. No evento **1**, MR notifica CCC que a fatia $F0$ foi liberada. Em **2**, CCC envia um sinal de reconhecimento para MR. Em **3**, CCC requisita os serviços do escalonador e envia o identificador da configuração que terminou de executar, nesse caso $C0$. Em **4**, TAR é atualizada, ou seja, a fatia $F0$ recebe o valor que indica que ela está livre. Como o sistema só possui seis configurações, adotou-se a constante '7' para indicar que uma fatia está livre. O evento **5** representa o recebimento de pedidos de configuração provenientes do EC. Nesse caso, EC requisita a configuração de $C1$. Em **6**, CCC envia o sinal de reconhecimento para EC e armazena o identificador a configuração escalonada em BCE. No evento **7**, CCC requisita ao AC a configuração de $C1$ na fatia livre $F0$.

Para essa prototipação, os módulos EC e MR foram integrados ao CCC, provendo assim uma validação mais realista. Acompanhando os sinais que revelam o comportamento do EC, pode-se perceber que o mesmo cronograma obtido na validação do EC aparece aqui.

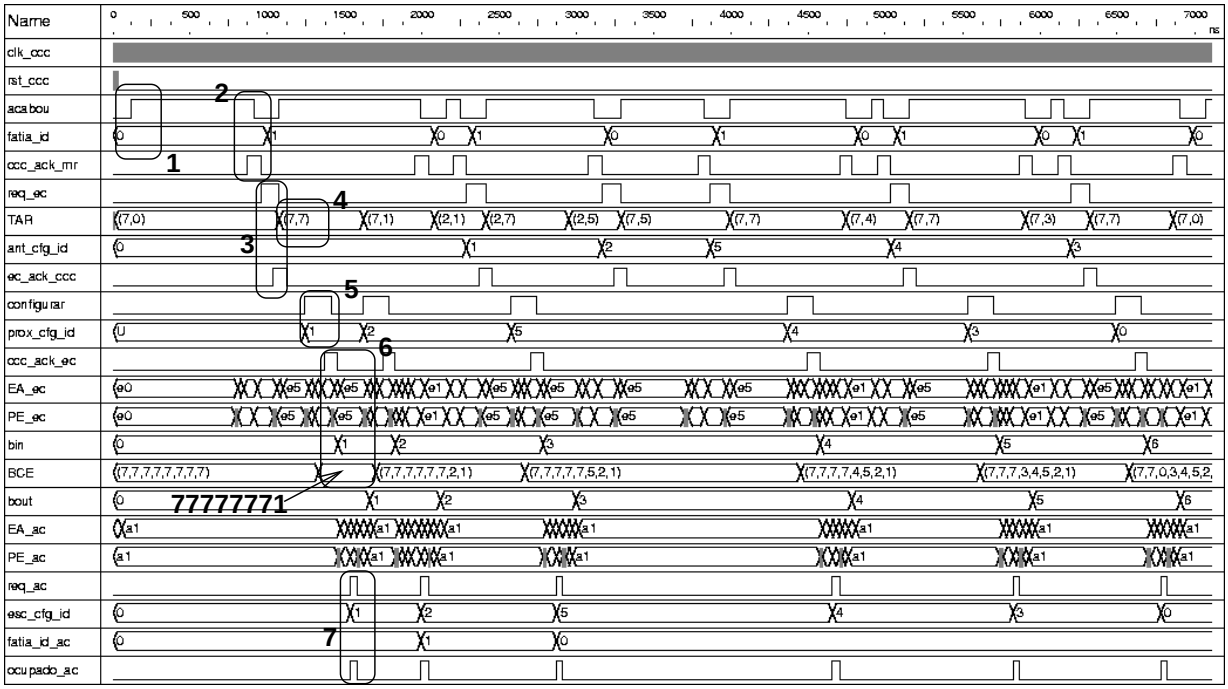

Figura 5.12: Forma de onda da simulação do módulo CCC – **1** *é sinalização de MR.* **2** *é o reconhecimento dessa sinalização.* **3** *é a requisição de EC.* **4** *é a atualização de TAR.* **5** *é a resposta da EC.* **6** *é a atualização de BCE.* **7** *é a requisição de AC.*

5.6 A Memória de Configurações

O tamanho da Memória de Configurações deve ser definido de acordo com o tamanho do sistema a ser implementado. Devido ao fato da implementação do RSCM utilizar dispositivos da família Virtex-II da Xilinx disponíveis na plataforma de prototipação V2MB1000 da Memec-Insight [MEM02], num primeiro instante pode-se notar a existência de três possibilidades para armazenamento utilizando os recursos dessa plataforma:

1. utilizar os blocos de memória RAM (BRAM) internos ao FPGA;
2. empregar a memória dinâmica (SDRAM [MIC03]) disponível na plataforma;
3. usar a memória estática (SRAM [TOS01]) disponível no módulo de expansão P160 [MEM02a] que acompanha a plataforma.

Após avaliar as capacidades de cada uma da três opções pode-se descartar a memória BRAM já que esta pode proporcionar no máximo 80kbytes para o dispositivo escolhido (40 blocos BRAM com capacidade de armazenamento de 18Kbits cada bloco) para armazenamento. Bitstreams parciais possuem tamanhos na ordem de 30kbytes dependendo, é claro, do tamanho do núcleo. Assim, o uso de BRAMs internas é insuficiente para desempenhar o papel de Memória de Configurações. Ao contrário da BRAM, as demais memórias apresentaram-se suficientes para desempenhar o papel da MC. A SDRAM pode armazenar

32Mbytes enquanto que a SRAM pode armazenar 1Mbytes. Essas duas alternativas são tratadas a seguir.

5.6.1 O controlador de acesso à memória externa

Para utilizar uma das memórias (SRAM ou SDRAM) é necessário implementar controladores que provejam acesso eficiente e simplificado às mesmas. Primeiramente, desenvolveu-se um controlador para memória SRAM. Como essa memória possui uma interface simples, porém assíncrona, o módulo de controle de acesso à memória SRAM constitui-se de uma máquina de transição de estados, cujo principal objetivo é respeitar às restrições temporais impostas pela SRAM. Na máquina de transição de estados apresentada na Figura 5.13 expõem-se a funcionalidade do controlador de acesso à SRAM.

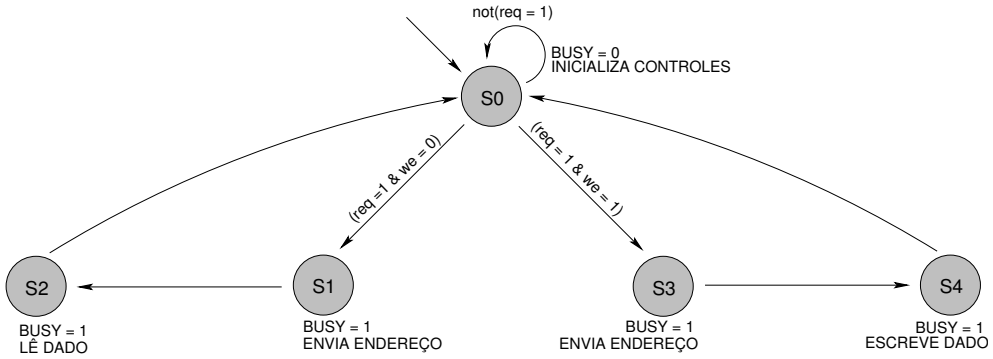

Figura 5.13: Máquina de estados do controlador de acesso à SRAM – *Depois da requisição para acessar a SRAM, uma leitura é realizada através da ramificação esquerda, ou uma escrita é realizada através da ramificação direita.*

O estado **S0**, responsabiliza-se por aguardar a requisição de serviços da memória e por detectar o tipo de serviço, ou seja se é uma leitura ou uma escrita. Os estados **S1** e **S2** realizam a leitura da memória e os estados **S3** e **S4** realizam uma escrita. Um cliente para essa memória deve se comportar da seguinte forma: requisitar o serviço desejado, setando *req*, *we* e *endereco* adequadamente, até que o controlador notifique que a memória está ocupada através do sinal *busy*. Em seguida, deve-se aguardar até que a memória desocupe-se para capturar o dados em caso de leitura ou, certificar-se que o dado foi armazenado, em caso de escrita. O endereçamento possui largura de 18 bits e os dados, 32 bits de largura. O controlador foi desenvolvido para receber um relógio de operação de 25MHz ou próximo deste. O sinal *we* seleciona o tipo de operação, sendo a escrita ativa em $we = 1$.

Também estudou-se um controlador de acesso à memória SDRAM. Adaptou-se um controlador disponível no site *http://www.opencores.org* para acesso à SDRAM. Através de simulação e prototipação pôde-se verificar o correto acesso à memória de configurações. Na Tabela 5.8 apresenta-se um comparativo entre as duas alternativas de implementação da MC. Nela percebe-se que o controlador de acesso à memória SDRAM é uma ordem de

grandeza maior, o que sugere a necessidade de analisar compromissos entre capacidade de armazenamento, velocidade de acesso e área ocupada pelo controlador. A memória SDRAM é síncrona e opera a uma freqüência máxima de operação de 133MHz, mas possui uma interface complexa, onde as operações realizadas baseiam-se em palavras de programação que determinam o comportamento da memória. A memória SRAM possui interface simples, mas tempo de acesso alto.

Tabela 5.8: Quadro comparativo dos controladores de acesso a memórias SRAM e SDRAM – *As duas alternativas para implementar a MC utilizando a plataforma de prototipação V2MB1000 da Memec-Insight [MEM02].*

Característica	**SRAM**	**SDRAM**
Tamanho da memória	1Mbyte	32Mbyte
Tempo de acesso	90ns	7,5ns
Tamanho do controlador*	0,39%	3,95%
Freqüência do controlador	25Mhz	100MHz

*Percentual de fatias ocupadas do dispositivo XC2V1000. Um bloco lógico configurável de um dispositivo da família VirtexII é composto por duas fatias.

A escolha para este trabalho recaiu sobre o controlador de SRAM, pois a dificuldade para integrar o controlador SDRAM comprometeria o cronograma de desenvolvimento deste trabalho.

5.6.2 A Interface com a Porta Serial

A necessidade de inicializar a Memória de Configurações com os arquivos parciais determina o desenvolvimento de um módulo que se responsabiliza por essa tarefa. O módulo Porta Serial é usado para carregar a memória de configurações com bitstreams parciais a partir de um computador hospedeiro. A lógica de controle desse sub-módulo não é apresentada aqui porque embora esse módulo faça parte do controlador RSCM, seu comportamento não influi na principal atividade do sistema, o controle de configurações em SDRs. Mais detalhes sobre este módulo podem ser obtidos em *hhttp://www.inf.pucrs.br/~gaph.*

A Porta Serial possui interface com um sistema composto por um módulo de hardware e um programa executável, cuja interface gráfica é ilustrada na Figura 5.14. O módulo de hardware, denominado *serialinterface*, é instanciado no sistema que reside no FPGA, enquanto que o software executa em um computador hospedeiro. Através deles, é possível transmitir dados entre o hospedeiro e o FPGA. A Porta serial do RSCM instancia o *serialinterface* e define um protocolo próprio para acesso à memória. Através desse protocolo, o usuário pode ler e escrever na memória bitstreams parciais. Uma seqüência de controle

deve ser enviada através do software para definir a operação a ser realizada. Esta seqüência é composta por três campos:

- 1 byte que define o tipo de operação (00 representa uma leitura e 01 uma escrita);
- 2 bytes informando o número de palavras de 32 bits a serem lidas ou escritas;
- 3 bytes informando o endereço inicial para leitura/escrita.

A seqüência hexadecimal ***00 000A 000003*** comanda a ***leitura*** de ***10*** palavras de 32 bits da Memória de Configurações a partir da posição ***3*** da memória. No caso da escrita, após enviar a palavra iniciada por 01 o usuário deve enviar todas as palavras de 32 bits indicadas no segundo campo, caso contrário a operação não será finalizada.

A interface com PS foi validada funcionalmente e prototipada em FPGA. Ambas tarefas apresentaram os resultados esperados. Na Figura 5.14, apresenta-se um exemplo da operação do módulo PS e sua interação com o software que envia dados do computador hospedeiro para a porta serial da plataforma. Esse software foi desenvolvido por bolsistas do grupo de pesquisa GAPH (Grupo de Apoio ao Projeto de Hardware), onde esse trabalho é realizado.

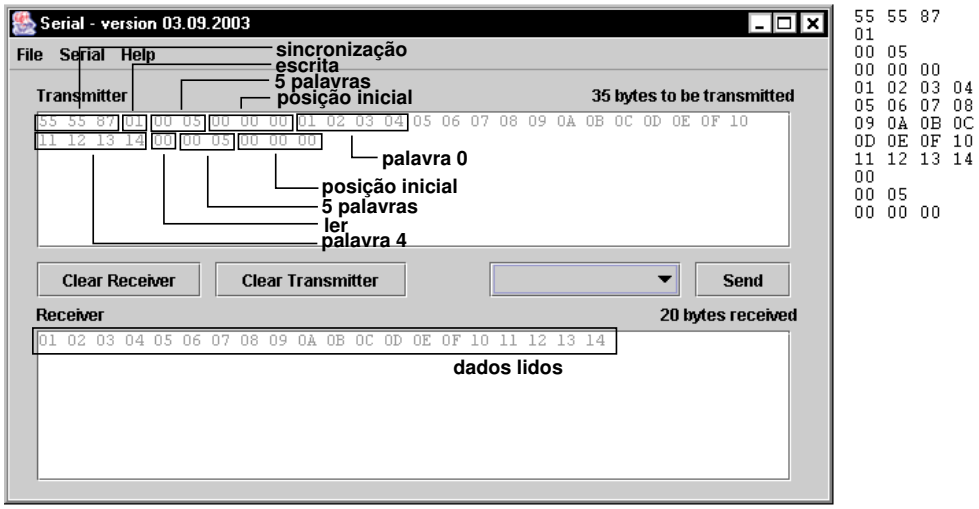

Figura 5.14: Tela capturada do software de interface com a porta serial da plataforma – *De acordo com a listagem ao lado da tela capturada, foram enviados para PS uma palavra de sincronização da interface serial, a seqüência para escrever cinco palavras na memória, a partir da posição zero, Em seguida as cinco palavras foram enviadas. Para verificar a correta operação de PS, foi também enviada uma seqüência para leitura dos dados escritos.*

5.6.3 O Multiplexador de acesso à memória de configurações

A memória de configurações pode ser acessada para inicialização do sistema pelo módulo PS e, também para configuração pelo módulo AC. Em virtude disso, o multiplexador para

acesso à memória foi desenvolvido. Um sinal seleciona qual dos módulos, PS ou AC, pode comunicar-se com o controlador de acesso à memória, $SRAMCTLR$. Como esse módulo constitui-se apenas de alguns poucos fios, não será aqui apresentada a sua validação.

5.7 Validação global do sistema RSCM

Depois de validar e prototipar cada um dos componentes do sistema RSCM individualmente, os módulos foram sistematicamente integrados em grupos, com a realização de validações das interfaces entre os mesmos. Dessa forma, pôde-se simular e prototipar o sistema por completo. Essa tarefa apresentou as respostas esperadas, parcialmente descritas nesta Seção.

Para o teste que segue foi adotado um sistema com três fatias e seis configurações. Foram utilizados arquivos hipotéticos compostos por oito palavras de configuração, sendo que as identificações da configuração e da fatia foram inseridas nas palavras de dados para facilitar os testes. Durante a prototipação do sistema, foram usados bitstreams reais, para evitar danificar o FPGA com bitstreams inválidos.

Na Figura 5.15 apresenta-se uma forma de onda que reflete o comportamento do sistema RSCM. Não serão abordados aqui os detalhes do funcionamento interno do sistema porque esses já foram tratados nas Subseções anteriores.

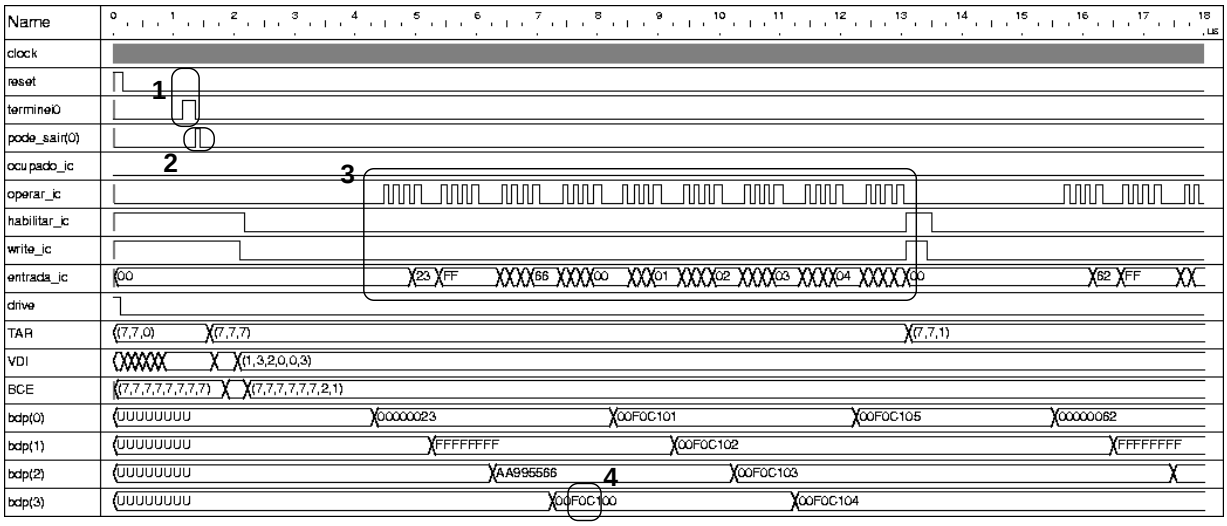

Figura 5.15: Forma de onda da simulação do sistema RSCM completo – **1** *é detecção da condição de reconfiguração.* **2** *é o reconhecimento da mesma.* **3** *é a reconfiguração do bitstream escalonado.* **4** *é uma amostra dos dados enviados para a interface de configuração.*

Existem quatro eventos a serem analisados nessa forma de onda. Em **1** a fatia zero notifica o RSCM que terminou sua execução. Como resposta, RSCM envia um sinal de

reconhecimento (i.e. *pode_sair*) para essa fatia, em **2**. Seguindo a operação do sistema, em **3**, é realizada uma reconfiguração coordenada por RSCM. Os sinais envolvidos fazem parte da interface com a porta de configuração do dispositivo, seja ela acessada através da ICAP ou da interface SelectMAP. Todas a nove palavras são enviadas para a interface, corretamente. De acordo com as informações contidas nas palavras de configurações hipotéticas, em **4**, pode-se perceber que o bitstream $C1$ está sendo configurado na fatia $F0$. Outras estruturas internas do sistema, tais como TAR e BCE são apresentadas na Figura 5.15 para que o leitor possa acompanhar o andamento do processo. A linha BCE apresenta as configurações escalonadas pelo escalonador. Acompanhando TAR, tem-se a ocupação das fatias gerenciada por CCC. Em BDP têm-se as palavras lidas da MC por AC e enviadas para IC. VDI expressa o número de dependências pendentes de cada uma das configurações, manipulada por EC.

Em todos os testes realizados foi utilizado um relógio de operação com freqüência de 24MHz. Até então, não se abordou a possibilidade de aumentar a freqüência de operação porque num primeiro instante não se objetiva alcançar o máximo desempenho do sistema como um todo, mas sim a operacionalidade do mesmo. Mais uma vez, antecipa-se uma tarefa como sugestão de trabalho futuro, qual seja, a realização de estudos para aumentar o desempenho do sistema RSCM.

5.8 Recursos de apoio à geração de controladores

Durante o desenvolvimento do RSCM foram desenvolvidos três recursos que auxiliam o desenvolvedor a utilizar o sistema para o controle de configurações em SDRs. O primeiro é a parametrizabilidade do código VHDL do RSCM. Os segundo e terceiro são programas para gerar o conteúdo da Memória de Configurações.

5.8.1 Parametrização do RSCM

Como mencionado anteriormente, o sistema RSCM deve ser parametrizado antes da síntese do SDR através do pacote *rscm_pkg.vhd*. Para gerar esse arquivo o desenvolvedor pode utilizar a ferramenta ***GeradorPkg***. Desenvolvida em linguagem C++, esta ferramenta não possui interface gráfica e comunica-se com o usuário através de linhas de comando, principalmente, através de um arquivo que contém informações sobre o escalonamento de configurações do sistema. Esse arquivo contém uma lista de vértices seguidos por suas ligações, representando um grafo tal qual o ilustrado na Figura 4.3 do Capítulo 4.

A parametrização do sistema RSCM, ou seja, definição do escalonamento a ser adotado e do tamanho das estruturas internas se dá apenas em tempo de projeto. Em virtude disso, para as modificações serem aplicadas, necessita-se de uma síntese do sistema. Idealmente a parametrização deveria poder ser realizada em tempo de execução, mas até o presente momento tal estratégia não foi adotada, pois complicaria o desenvolvimento do sistema

tornando-o inviável para o tempo de desenvolvimento disponível.

5.8.2 Gerando o conteúdo da Memória de Configurações

Duas outras ferramentas foram desenvolvidas para gerar os dados que devem ser escritos na MC. O usuário pode criar os arquivos de configuração no formato hexadecimal a partir dos arquivos de configuração parciais em formato RBT, utilizando a ferramenta ***bin2hex***. Tal ferramenta foi desenvolvida em linguagem C++ e também não possui interface gráfica. Basta ao usuário executar o programa em linha de comando e passar como parâmetro o arquivo de configuração RBT. A saída é um arquivo com o nome *resultado.txt*.

A outra ferramenta desenvolvida gera o conteúdo da memória de acordo com a estrutura ilustrada na Figura 5.5 apresentada no Capítulo anterior. Tal ferramenta denomina-se ***GeradorMC*** e foi desenvolvida em linguagem Java. GeradorMC não possui interface gráfica e recebe como entrada um arquivo que contém a lista ordenada de todas as configurações parciais do sistema. Como saída, é gerado um arquivo com todo o conteúdo a ser escrito na memória, com área de ponteiros e dados de configuração. Esse arquivo deve ser utilizado como entrada no programa da porta serial apresentado na Figura 5.14.

No próximo Capítulo, apresenta-se estudos de casos do sistema RSCM que corroboram para a certificação de seu funcionamento. Também apresenta-se os resultados deste trabalho, e um conjunto de possíveis desdobramentos futuros.

Capítulo 6

Estudos de Caso

Como anteriormente citado, SDRs são sistemas computacionais flexíveis na medida que seu comportamento pode ser alterado após fabricação. Além disso, podem ser implementados em uma área reduzida, proporciona uma economia de recursos já que partes desse sistema podem ser configuradas dinamicamente sob demanda da aplicação. Um exemplo típico de sistemas dessa natureza é apresentado na Seção 6.2, onde um processador com um conjunto de instruções variável é implementado. Esse exemplo tem como principal finalidade enfatizar a importância do emprego de técnicas RTR no desenvolvimento de sistemas de hardware.

Neste Capítulo, apresenta-se o estudo de caso elementar usado para validar o sistema RSCM, na Seção 6.1. Após a discussão do processador reconfigurável, na Seção 6.2, apresenta-se, na Seção 6.3, alguns dados quantitativos sobre o sistema RSCM implementado.

6.1 Estudo de caso 1: Projeto LEDs

O projeto LEDs é um sistema implementado para realizar a prova de conceito do controlador de configurações RSCM. Sua operação não realiza qualquer outra tarefa senão prover o acompanhamento passo a passo da operação do controlador de configurações. Os detalhes de sua estrutura são tratados a seguir.

6.1.1 Estrutura do sistema

Na Figura 6.1 apresenta-se a estrutura do sistema LEDs. Este é composto por cinco módulos, dos quais três são fixos, não reconfiguráveis, enquanto os outros dois são reconfiguráveis. Os três módulos fixos são:

- **Interface com usuário**: módulo que apresenta nos *displays* de sete segmentos da plataforma o identificador da configuração residente em cada uma das duas áreas

reconfiguráveis.

- **Decodificador**: módulo que interpreta o identificador dos bitstreams contidos nas áreas reconfiguráveis e provê a passagem de sinais entre as áreas reconfiguráveis e o controlador RSCM. Este módulo é responsável por gerar informações para a Interface com o Usuário.
- **RSCM**: controlador de configurações do sistema

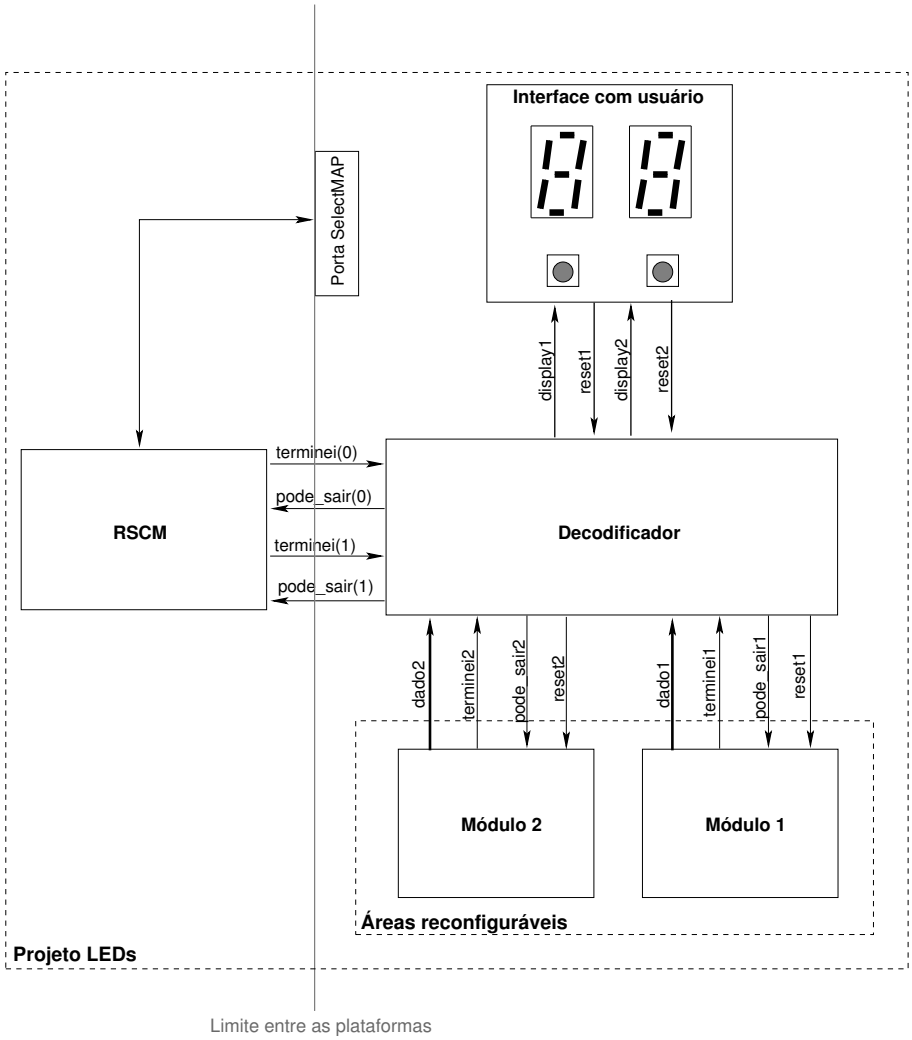

Figura 6.1: Estrutura do sistema LEDS – *O usuário pode acompanhar a execução do SDR através dos displays da plataforma XC2V1000. Os números apresentados são os identificadores dos bitstreams carregados em cada uma das duas fatias reconfiguráveis.*

Os outros dois módulos do sistema são representados pelas duas áreas reconfiguráveis, nas quais os bitstreams parciais do sistema são inseridos. A lógica de todas as configurações

do sistema é composta por uma máquina de estados que conta por um determinado período de tempo, aproximadamente 10s, enviando um identificador próprio para a área fixa. Após esse período, notifica que terminou sua execução e aguarda o reconhecimento do RSCM, transmitido através do Decodificador. Cada uma das cinco configurações implementadas possui seu próprio identificador. Durante a operação do sistema pode-se acompanhar a reconfiguração de bitstreams parciais, observando os displays, que indicam qual bitstream está em cada uma das áreas reconfiguráveis. Se a seqüência de reconfigurações se der de acordo com o cronograma de escalonamento escolhido e os bitstreams se comportarem corretamente, pode-se concluir que o controlador RSCM funcionou da forma para qual foi implementado.

6.1.2 Experimentos conduzidos

Foi implementado um sistema utilizando duas plataformas V2MB1000, a primeira delas apenas com o controlador RSCM, e a segunda com o restante dos módulos. Essas duas plataformas foram conectadas através de fios, que implementam a interface entre os sinais transmitidos entre o RSCM e as áreas reconfiguráveis. Além disso, os sinais de configuração gerado pelo RSCM foram conectados à interface de configuração SelectMAP da segunda plataforma.

Acompanhando a operação do sistema, pôde-se verificar o correto funcionamento do mesmo. A partir desta constatação conclui-se que o controlador de configuração RSCM comporta-se da forma esperada.

6.1.3 Resultados obtidos

Além de validar o RSCM, pôde-se obter estimativas do tempo gasto durante a reconfiguração do sistema LEDs, uma medida importante para definir os compromissos que validam a utilização de técnicas RTR, de acordo com o tempo gasto para reconfigurar o sistema.

Como citado por Wirthlin [WIR97], em sistemas reconfiguráveis dotados de um núcleo controlador de configurações, o tempo usado por este para modificar o sistema pode ou não contribuir para o tempo total de execução das tarefas para as quais o sistema foi originalmente concebido. Como os tempos de reconfiguração total práticos encontram-se hoje na ordem de ms [BRI02] e os parciais na ordem de μs, a operação do controlador de configurações é portanto crítica, podendo inviabilizar ou viabilizar o uso de técnicas RTR.

Configurações composta por 460 palavras de dados foram reconfiguradas no sistema com tempos na ordem de μs. A partir daí, foram obtidos os tempos de inicialização da reconfiguração e o tempo para enviar cada palavra à interface ICAP. O tempo de inicialização corresponde ao intervalo utilizado para buscar a primeira palavra da memória, já que as demais leituras são escondidas por sua sobreposição à tarefa de reconfiguração,

tal qual um pipeline. O tempo de inicialização obtido foi **884ns**. O tempo para enviar cada palavra obtida foi de **748ns**. Substituindo estes valores na Equação 6.1 do tempo de reconfiguração, obtém-se os valores correspondentes, específicos para o sistema RSCM em questão, conforme descrito pela Equação 6.2.

$$T_{rec} = T_{init} + (Num_{palavras} \times T_{palavra}) \tag{6.1}$$

$$T_{rec} = 884ns + (Num_{palavras} \times 748ns) \tag{6.2}$$

Este mesmo procedimento foi realizado com informações obtidas através de uma reconfiguração utilizando o software Impact, no modo Paralelo. Observou-se que o tempo gasto para configurar cada uma das palavras do bitstream é **8,44μs**. O tempo de inicialização é de aproximadamente **160ms**. Dessa forma, pôde-se definir uma equação específica para os sistemas configurados através do software Impact, a Equação 6.3. O tempo de inicialização utilizando o software Impact é grande por causa da baixa freqüência utilizada, como visto na Figura 5.8(página 80).

$$T_{rec} = 160ms + (Num_{palavras} \times 8,44\mu s) \tag{6.3}$$

Com as Equações 6.3 e 6.2 foi criado o gráfico apresentado na Figura 6.2. Este compara os tempos de reconfiguração, utilizando os dois métodos citados, de acordo com o tamanho do arquivo de configuração. Um arquivo de configuração total para dispositivos da família VirtexII da Xilinx possui 127.581 palavras de 32 bits [XIL02c]. Como citado, os dispositivos dessa família são particionados em colunas de elementos e estas, por sua vez, são divididas em quadros verticais. Um quadro é a unidade atômica de configuração que necessita de 153 palavras de configuração para o dispositivo XC2V1000. Este é número mínimo de palavras de um bitstream parcial para o dispositivo usado. Infelizmente, nenhuma ferramenta para geração de arquivos parciais do conhecimento do autor habilita gerar um arquivo com esse tamanho mínimo. Os menores arquivos gerados, até o presente momento, possuem 460 palavras, suficientes para configurar três quadros.

No gráfico da Figura 6.2 nota-se que o tempo de reconfiguração utilizando o Autoconfigurador do sistema RSCM é sempre significativamente menor que aquele obtido utilizando o software Impact.

6.2 Estudo de caso 2: Projeto R8R

Esta Seção apresenta um estudo de caso mais elaborado que o anterior, enfatizando a importância do emprego de técnicas RTR em sistemas programáveis. O sistema que compõe esse estudo denomina-se R8R (i.e. R8 Reconfigurável), sendo um processador dinâmica e parcialmente reconfigurável.

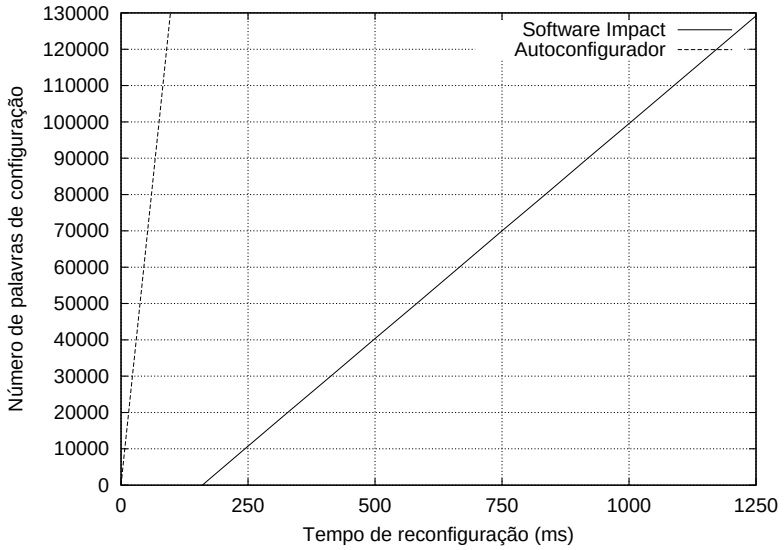

Figura 6.2: Relação tamanho do arquivo de configuração versus tempo de reconfiguração – *O tamanho do arquivo é dado por seu número de palavras de 32 bits. O tempo é representado em ms.*

6.2.1 O processador R8

O processador R8 [MOR03a] faz parte de uma família de processadores, concebida com a finalidade de dar suporte ao ensino de conceitos de arquitetura e organização de computadores a nível de graduação e pósgraduação [CAL01]. Este processador é uma máquina Von Neuman, com memória de dados e instruções conjunta. Ele possui uma arquitetura load-store, onde as operações lógico/aritméticas são executadas entre registradores, e as operações de acesso à memória só executam ou uma leitura ou uma escrita de uma posição de memória. Devido à característica load/store, o processador deve ter um conjunto de registradores de trabalho, para reduzir o número de acessos à memória.

Todas as instruções do processador R8 possuem exatamente o mesmo tamanho, e ocupam 1 palavra de memória. Cada instrução contém o código da operação e o(s) operando(s), caso exista(m). R8 é um processador de 16 bits pois manipula dados e endereços com uma largura de 16 bits. A execução de cada instrução é realizada em 3 ou 4 ciclos de relógio, ou seja, todo programa executado pelo processador possui CPI (i.e. Ciclos Por Instrução) entre 3 e 4. O processador possui um banco de 16 registradores de uso geral (R0 a R15) e 4 flags de estado: negativo, zero, carry, overflow. Cada endereço de memória corresponde a um identificador de uma posição onde residem 16 bits de conteúdo (i.e. o endereçamento de memória é a palavra). Dessa forma, o processador R8 é praticamente uma máquina RISC, faltando, contudo algumas características que existem em qualquer máquina RISC, tal como pipelines.

O conjunto de instruções do processador realiza:

- operações lógicas e aritméticas binárias (e.g. soma, subtração, E, OU, OU exclusivo);
- operações lógicas e aritméticas com constantes curtas (e.g. soma, subtração);
- operações unárias (e.g. deslocamento à direita, à esquerda e inversão);
- carga de metade de um registrador com uma constante;
- inicialização do apontador de pilha e retorno de subrotina;
- operação vazia (e.g. NOP ou *no operation*);
- suspensão da execução de instruções (e.g. HALT);
- leitura de posição de memória para um registrador;
- armazenamento de dado de um registrador em uma posição de memória;
- Saltos e chamadas de sub-rotina com endereçamento relativo com deslocamento curto ou longo (contido em um registrador) e endereçamento absoluto (a registrador);
- Inserção e remoção de valores no/do topo da pilha de dados.

O processador R8 conta ainda com um conjunto de registradores de controle de 16 bits, em acréscimo aos 16 registradores de uso geral. O registrador IR (*instruction register*) armazena o código de operação da instrução atual e o(s) operando(s) da mesma. O registrador PC (*program counter*) é o contador de programa. O SP (*stack pointer*) armazena o endereço do topo da pilha e controla a chamada e retorno de sub-rotinas. O banco de registradores do processador possui uma porta de escrita e duas de leitura, que habilitam a realização de uma escrita e duas leituras de forma simultâneas.

6.2.2 O projeto R8R

O projeto emprega conceitos de processadores que contém um conjunto de instruções dinâmico, demoninados *Dynamic Instruction Set Computers* ou DISCs, tratados por Wirthlin e Hutchings, em [WIR95, WIR95a]. Tal projeto propõe o desenvolvimento de um processador dinâmica e parcialmente reconfigurável, baseado na estrutura do processador R8 e em uma versão simplificada do controlador de configurações RSCM proposto. O principal objetivo desse sistema é possibilitar a implementação de um processador com um conjunto de instruções variável e expansível. Além das instruções padronizadas do processador R8, o R8R agrega outras instruções destinadas a controlar a execução de coprocessadores reconfiguráveis. Esses coprocessadores são configurados no sistema de acordo com a demanda definida pelo software do R8R, e podem executar as mais diferentes tarefas, tais como operações aritméticas complexas e entrada/saída de dados em altas taxas.

Na Figura 6.3 apresenta-se um diagrama de blocos do processador reconfigurável R8R. O sistema compõe-se do processador R8R, da memória de programa e dados, do controlador de configurações RSCM e dos coprocessadores dinâmica e parcialmente reconfiguráveis. A interface entre esses módulos é composta pelos sinais apresentados na Tabela 6.1.

Os coprocessadores são selecionados quando seu identificador é idêntico ao valor transmitido através do barramento *IOaddress*. Desta forma, todos os coprocessadores são

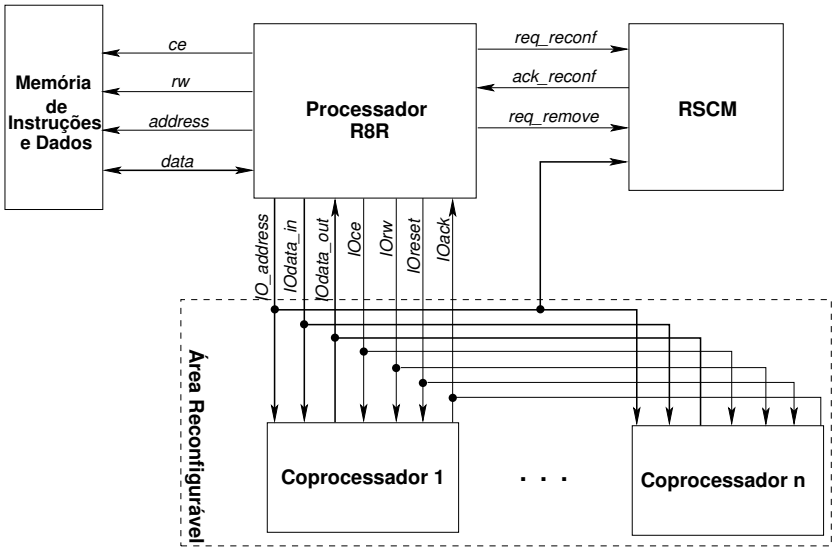

Figura 6.3: Estrutura do processador reconfigurável R8R – *Este processar possui interfaces com o controlador RSCM e com os coprocessadoes implementados em hardware. Estas interfaces são apresentadas na Tabela 6.1.*

Tabela 6.1: Sinais manipulados da interface da R8R – *Os sete primeiros sinais pertencem a interface do processador R8R com os coprocessadores. Os sinal IOaddress é transmitido também ao controlador RSCM, juntamente com os três outros sinais presentes nesta Tabela.*

Sinais	***Descrição***
$IOce$	sinal enviado do processador para o coprocessador, habilitando a operação do último.
$IOrw$	sinal enviado do processador para o coprocessador, selecionando o tipo de operação a ser realizada (1 para escrita e 0 para leitura).
$IOreset$	sinal enviado do processador para o coprocessador, visando resetar o último.
$IOack$	sinal enviado do coprocessador para o processador, visando sinalizar o término da operação do primeiro.
$IOdata\_in$	barramento de 16 bits comunicando dados do coprocessador ao processador.
$IOdata\_out$	barramento de 16 bits comunicando dados do processador ao coprocessador.
$IOaddress$	barramento de 8 bits comunicando endereço de acesso do processador a um coprocessador. Os coprocessadores utilizam esse barramento para verificar se os dados e sinais de controle enviados são endereçados a ele. O controlador de configurações utiliza esse barramento para identificar o coprocessador a ser configurado.
req_reconf	sinal enviado do processador para o controlador de configurações RSCM, requisitando a configuração do coprocessador identificado em $IOaddress$.
ack_reconf	sinal enviado do controlador de configurações para o processador, sinalizando a término da execução do controlador.
req_remove	sinal enviado do processador para o controlador de configurações requisitando desconfiguração do coprocessador identificado em $IOaddress$.

relocáveis em qualquer área reconfigurável do sistema. O processador não necessita saber em qual área reconfigurável encontra-se o coprocessador desejado.

Diferenças entre o processador R8 e o processador R8R

Para implementar o sistema R8R foi necessário adequar o processador R8 original, inserindo interfaces com os coprocessadores reconfiguráveis (16 bits) e com o controlador de configurações RSCM (10 bits). Ainda, criaram-se cinco novas instruções para o processador que habilitam seleção, inicialização, sinalização, leitura e escrita nos coprocessadores reconfiguráveis (SELR, INITR, DISR, RDR e WRR). A Tabela 6.2 apresenta as instruções adicionadas ao conjunto de instruções do processador R8 adaptando-o ao sistema R8R.

Tabela 6.2: Intruções adicionadas ao processador R8 para gerar o processador R8R – *As duas primeiras instruções, SELR e DISR, geram sinais para a operação do controlador RSCM. Estas são bloqueantes, e aguardam o término da operação do RSCM, sinalizada por ack_reconf. As demais instruções dedicam-se a manipular os coprocessadores.*

Instrução	***Descrição***
SELR *imed8*	Instrução que seleciona para uso o coprocessador identificado por *imed8*. Se o coprocessador não estiver configurado, o RSCM o configura e gera o *ack_reconf*. Caso contrário RSCM realiza a reconfiguração do coprocessador de forma transparente ao processador R8R. Como *imed8* é armazenado em **RRECONF**, instruções subseqüentes de acesso a um coprocessador implicitamente endereçam o mesmo coprocessador até a execução da próxima instrução SELR.
DISR *imed8*	Informa ao controlador RSCM que o coprocessador identificado por *IOaddress* pode ser removido do sistema. *IOaddress* recebe imed8 e o sinal *req_remove* é ativado. Após realizar suas tarefas, o controlador ativa *ack_reconf* notificando o processador quanto ao término de sua operação.
INITR *imed8*	Inicializa um coprocessador identificado pelo operando da instrução, passado ao barramento *IOaddress*. O sinal *IOreset* é ativado, solicitando a inicialização do coprocessador. O processador deve aguardar o sinal *IOack* vindo do coprocessador.
WRR *Rs1, Rs2*	Escreve dados de 16 bits no coprocessador. Essa instrução habilita o sinal *IOce* para acesso ao coprocessador identificado por **RRECONF** (um registrador interno do R8R que contém a identificação do coprocessador selecionado pela última instrução SELR executada), e atribui 1 ao sinal *IOrw* , selecionando uma operação de escrita. O conteúdo do registrador **Rs1** é enviado para o barramento *IOdata_out*. A seguir, o processador aguarda o sinal *IOack* e escreve o conteúdo de **Rs2** no barramento *IOdata_out*. Novamente, o processador aguarda que o sinal *IOack*.
RDR *Rs1, Rt1*	Lê dados de 16 bits enviados pelo coprocessador. Tal instrução habilita *IOce* e atribui 1 ao sinal *IOrw* , selecionando uma operação de escrita. O conteúdo do registrador **Rs1** é enviado para o barramento *IOdata_out*, aguarda Em seguida, seta *IOrw* para zero, selecionando operação de leitura do processador. O coprocessador deve escrever sua saída no barramento *IOdata_in* do processador, que espera *IOack* e escreve o dado lido no registrador **Rt1**.

Como citado acima, as instruções de leitura e escrita no coprocessador, sempre iniciam-se pela escrita de uma palavra de 16bits. Esta palavra pode ser utilizada como um controle da operação do coprocessador selecionado.

Modificações realizadas no controlador de configurações RSCM

Em virtude do escalonamento e a monitoração de configurações ser tarefa realizada pelo processador R8R em software, os módulos que realizam estas tarefas no sistema RSCM são desnecessários e foram removidos. O processador R8R possui interface com o Controle Central de Configurações (CCC) do sistema RSCM. O projeto R8R implementa um sistema dinâmica e parcialmente reconfigurável com escalonamento dinâmico de configurações determinado pelo fluxo de execução do programa executado pelo processador. Na Figura 6.4 ilustra-se estrutura personalizada do controlador de configurações RSCM para o projeto R8R.

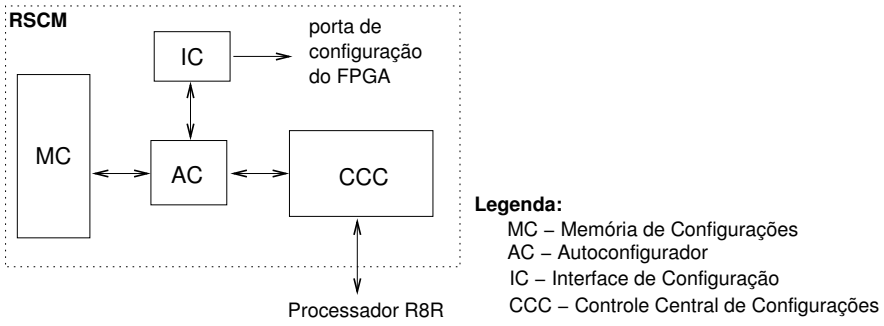

Figura 6.4: RSCM adaptado para o projeto R8R – *Os módulos Escalonador de Configurações e Monitor de Reconfiguração foram removidos do RSCM. No projeto R8R, é o software executando no processador de determina quando e qual coprocessador deve ser escalonado.*

Comparando a Figura 6.4 com a Figura 4.1, apresentada no Capítulo 4, nota-se que o Monitor de Reconfiguração (MR) e o Escalonador de Configurações (EC) foram substituídos pelo processador reconfigurável. As interfaces do Controle Central de Configurações (CCC) sofreram mudança. Por isso, o comportamento e a estrutura desse módulo teve de sofrer modificações. Na Figura 6.5, apresenta-se a nova máquina de estados do Controle de Central de Configurações adaptado para o sistema R8R.

Pode-se notar que a máquina da Figura 4.1 é externamente simples porque foi projetada para um caso simples de aplicação, no caso um SDR que possui apenas uma área reconfigurável. Dessa forma, muitas das estruturas internas do CCC foram removidas tal como o Buffer de Configurações Escalonadas. Para essa implementação, não é necessário utilizar duas máquinas de estados como na versão original do CCC.

A máquina da Figura 4.1 possui cinco estados. No estado **E0**, CCC inicializa os sinais de reconhecimento de requisições a partir do CCC (i.e. sinais req_reconf e req_remove) e o sinal que requisita serviços do Autoconfigurador AC (i.e. req_ac). No estado **E1**, CCC aguarda o processador R8 requisitar os serviços do controlador RSCM. Quando isso acontece, CCC identifica qual a natureza da requisição, ou seja, se é um pedido de reconfiguração

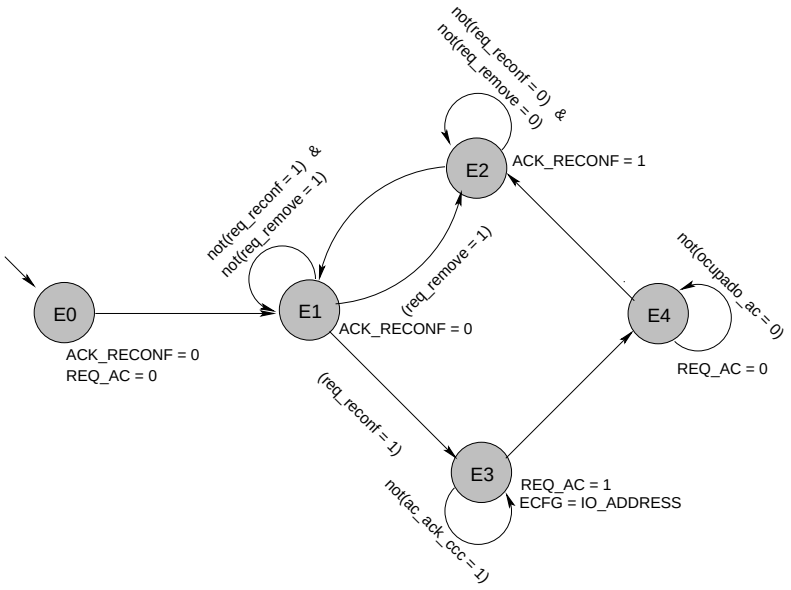

Figura 6.5: Nova máquina de estados do CCC adaptado para o projeto R8R. – *Esta máquina, aguarda a requisição vinda do processador R8R. CCC verifica a necessidade de reconfigurar o coprocessador selecionado e o reconfigura, se necessário. Em seguida, avisa R8R que terminou a operação.*

ou uma notificação que a área reconfigurável pode ser liberada. A tarefa de liberar a área não faz sentido no sistema com uma área reconfigurável apenas, mas deverá ser utilizado nas próximas versões do processador R8R com diversas áreas reconfiguráveis. No estado **E3**, CCC requisita os serviços do Autoconfigurador e, em **E4** aguarda AC terminar a tarefa de reconfiguração. Finalmente, no estado **E2**, CCC envia o sinal à R8R, notificando que a tarefa solicitada foi concluída.

6.2.3 Experimentos conduzidos

Para testar o funcionamento do sistema R8R foi utilizado o programa descrito no Apêndice A.1. Pôde-se verificar o correto funcionamento do sistema, já que a operação do programa apresentou as respostas corretas.

Além de desenvolver os módulos apresentados na Figura 6.3 foram desenvolvidos cinco coprocessadores, listados abaixo.

- **2-avg**: coprocessador que calcula a média entre dois valores;
- **4-avg**: coprocessador que calcula a média entre quatro valores;
- **sqrt**: coprocessador que calcula a raiz quadrada de um valor de 32bits e apresenta uma resposta de 16bits;

- **multi**: coprocessador que executa a multiplicação de dois valores de 16 bits, gerando uma resposta em 32 bits;
- **div**: coprocessador que executa a divisão entre dois valores de 16 bits, gerando uma resposta em 32 bits.

6.2.4 Comparação software versus hardware reconfigurável

Para cada um dos coprocessadores desenvolvidos em hardware foi desenvolvida uma versão em software. Executando programas que utilizam esses recursos por determinado tempo, pôde-se obter um comparativo entre o desempenho global do sistema quando se utiliza apenas rotinas desenvolvidas em software e, quando se utiliza coprocessadores implementados em hardware. O código dos programas que realizam as mesmas tarefas dos coprocessadores ***multi***, ***div*** e ***sqrt*** apresenta-se nos Apêndices A.2, A.3 e A.4 respectivamente.

Como o R8R possui fatias de tamanho único, mesmo os coprocessadores possuindo áreas diferentes, todos os bitstreams gerados possuem o mesmo tamanho. O tempo de reconfiguração de cada um é de aproximadamente 10ms. Este valor poderá ser visto nas próximas Figuras, pois é adicionado ao tempo de execução das tarefas utilizando os coprocessadores.

No caso do coprocessador ***multi***, observou-se que programas que realizam mais que **750** operações de multiplicação possuem um melhor desempenho quando comparado as rotinas implementadas em software. Esse resultado pode ser visto na Figura 6.6.

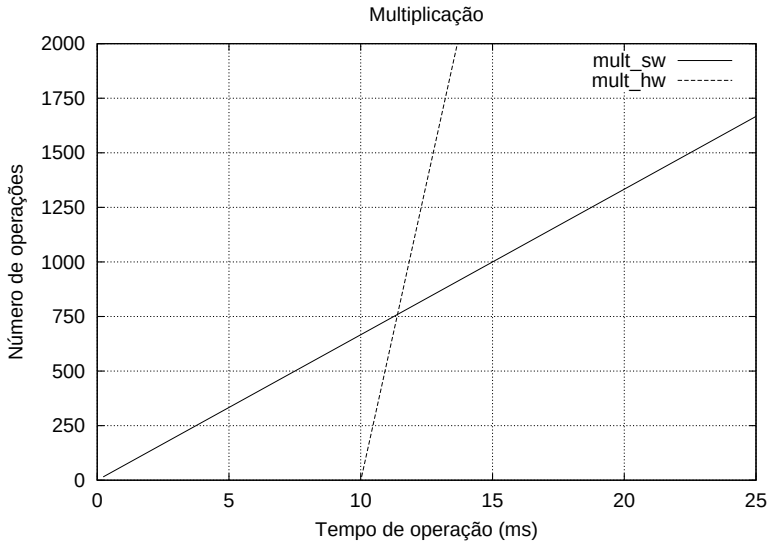

Figura 6.6: Implementação do coprocessador ***multi*** em hardware x software – *A partir de 750 operações o coprocessador* ***multi*** *implementado em hardware apresenta um melhor desempenho.*

O mesmo experimento foi realizado para os coprocessadores ***div*** e ***sqrt***. Programas que realizam mais de **260** operações de divisão possuem um melhor desempenho quando utilizam o processador ***div*** em vez da rotina implementada em software, como pode ser visto na Figura 6.7. Na Figura 6.8, nota-se que apenas **200** operações de extração da raiz quadrada já validam o uso do coprocessador ***sqrt*** em contraproposta à implementação em software.

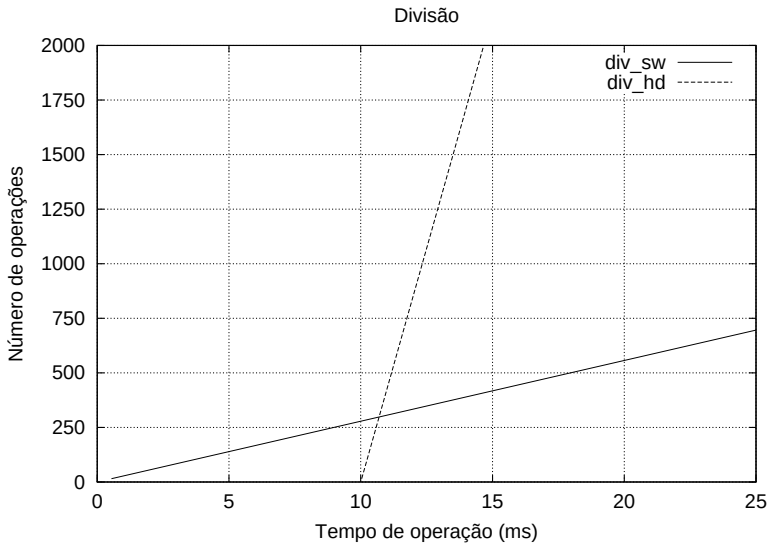

Figura 6.7: Implementação do coprocessador ***div*** em hardware x software – *A partir de 260 operações o coprocessador* ***div*** *implementado em hardware apresenta um melhor desempenho.*

A curvas referentes a operação dos coprocessadores implementados em hardware representam uma abordagem do pior caso. Isso porque se admite que o sistema possui uma única área reconfigurável. Se o sistema possuir diversas áreas reconfiguráveis, o tempo de reconfiguração pode ser ocultado.

Para melhor entender a importância do uso dos coprocessadores em contraproposta às rotinas em software pode-se imaginar um sistema de software que aplica uma série de filtros em imagens. Se esses filtros aplicam multiplicações, divisões, médias de valores e raízes quadradas constantemente, muito tempo poderá ser ganho utilizando os coprocessadores de alto desempenho, ainda mais se esses filtros são aplicados a diversas imagens ou se as mesmas forem grandes.

Duas das mais importantes vantagens do processador implementado utilizando técnicas de reconfiguração dinâmica e parcial são o aumento da densidade funcional e da flexibilidade do sistema, que pode ser modificado facilmente, através da adição de outros coprocessadores.

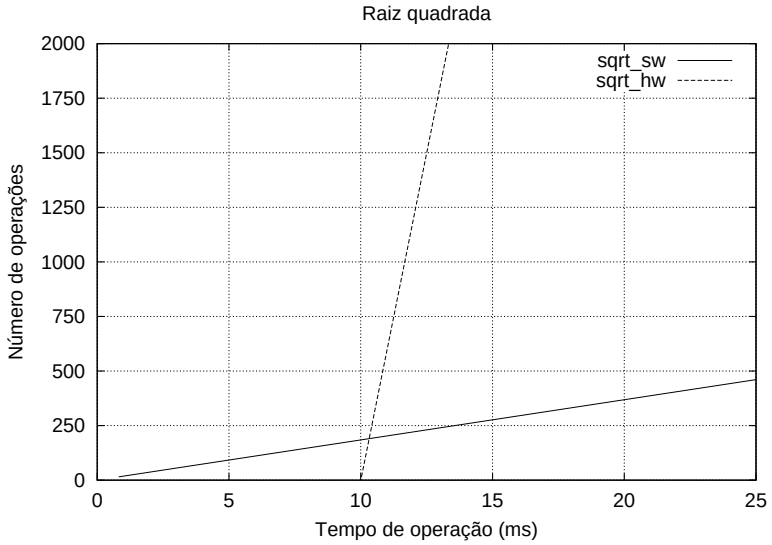

Figura 6.8: Implementação do coprocessador ***sqrt*** em hardware x software – *A partir de 200 operações o coprocessador* ***sqrt*** *implementado em hardware apresenta um melhor desempenho.*

6.3 Análise quantitativa do sistema RSCM

Nessa Seção apresenta-se uma análise quantitativa relacionada ao sistema RSCM proposto e implementado. As informações aqui expostas foram capturadas a partir da síntese lógica do sistema utilizando a ferramenta Leonardo Spectrum chamada a partir do ambiente ISE, fornecido pela Xilinx. Os valores reais foram obtidos após a síntese física. Na Tabela 6.3, apresentam-se informações referentes ao RSCM implementado em um dispositivo XC2V1000 da família VirtexII da Xilinx. Esse controlador opera com freqüência de 24MHz.

Tabela 6.3: Análise quantitativa do sistema RSCM – *Nesta Tabela, a área percentual é tomada apenas com relação ao número de LUTs do sistema.*

Módulo	*Número de LUTs*	*Número de DFFs*	*Área percentual*	*FSMs*
IC	0	0	0.00%	0
MR	16	11	0.16%	1
EC	83	34	0.81%	1
CCC	106	67	1.23%	2
AC	268	106	2.62%	2
MC	413	260	4.03%	3
RSCM estimado	**914**	**478**	**8,89%**	**9**
RSCM real	**754**	**395**	**7%**	**9**

Na Tabela 6.4, apresentam-se tempos de configuração de dispositivos XC2V1000, nos diferentes modos. Todos os valores apresentados correspondem ao tempo gasto para configurar o dispositivo totalmente, enviando todas as 127.581 palavras de configurações, seja no modo paralelo (i.e. 8 bits) ou serial. Com exceção do tempo relacionado à operação do

Autoconfigurador do sistema RSCM, os demais foram obtidos da referência [BRI02].

Tabela 6.4: Tempos de configuração de dispositivos XC2V1000 da família VirtexII – *Estes tempos revelam que o Atutoconfigurador de RSCM possui um desempenho 2 ordens de grandeza maior que as configurações. O menor desempenho em relação às configurações autônomas é causado pelo gargalo do controlador de memória SRAM utilizado no RSCM.*

Modo de configuração	***Freqüência*** (MHz)	***Tempo*** (ms)
Serial dependente	4,5	10200
Paralelo dependente	0,48	1510
Paralelo autônomo (RSCM)	**24**	**95,43**
Serial autônomo	60	62,2
Paralelo autônomo	60	7,5

Pode-se notar, nesta Tabela, que os maiores tempos de configuração foram obtidos quando o dispositivo é configurado no modo dependente. Isso se dá devido ao fato destas configurações serem realizadas através de cabos que transmitem dados de um processador hospedeiro para a porta de configuração do dispositivo em baixa velocidade. No modo autônomo, os dados foram transmitidos através de barramentos internos da plataforma, a partir de uma memória PROM. Quando a plataforma é inicializada, os dados são transferidos dessa PROM para a porta de configuração do dispositivo a partir de comandos gerados pelo próprio FPGA, se o modo autônomo é utilizado. Outro fator importante a ser notado é o fato da freqüência de configuração nos modos autônomos ser de 60MHZ. Segundo fabricante da PROM, a freqüência máxima de configuração é 33MHz. Contudo, foi possível configurar o dispositivo utilizando a PROM com uma freqüência superior aos 33MHz. A diferença de desempenho entre as configurações autônomas e aquela realizada pelo Autoconfigurador do RSCM é causada pelo gargalo da memória de configurações, cujo controlador de acesso opera a 24MHz apenas.

O Autoconfigurador do sistema RSCM, reconfigura o dispositivo no modo paralelo autônomo, e opera a uma freqüência quase três vezes menor que a opção que utiliza a PROM. Como a interface ICAP ainda não funcionou adequadamente, o RSCM não apresentou o melhor desempenho, mas superou em duas ordens de grandeza a alternativa usual de configurar parcialmente o dispositivo através de software operando no hospedeiro. Obviamente, não é possível, configurar o dispositivo totalmente de maneira interna, dessa forma o valor apresentado na Tabela corresponde apenas a uma estimativa obtida através da Equação 6.2. Por outro lado, a configuração utilizando a PROM não pode ser realizada parcialmente, ou pelo menos, não sem o emprego de um hardware intermediário entre PROM e FPGA. Sendo assim, essa opção não pode ser empregada em um sistema que controla configurações.

Outra observação decorre das características do controlador de acesso à Memória de Configurações do RSCM. Até o presente instante foi utilizado o controlador de SRAM, assíncrona e com ciclos de acesso longos. Para ler ou escrever um dado nesta memória,

ciclos de 90ns no mínimo são necessários, sendo que o controlador desenvolvido gera ciclos de 120ns. Um ganho de desempenho de configuração pode ser obtido se a memória SRAM for substituída no sistema pela memória SDRAM. Esta, por sua vez, é síncrona e opera a uma freqüência de 133MHz. Um comparativo destas duas memórias foi apresentado na Tabela 5.8.

No próximo Capítulo, apresentam-se as contribuições do trabalho, seguidas das conclusões obtidas e de sugestões para trabalho futuro.

Capítulo 7

Considerações finais

7.1 Resumo das contribuições

A principal contribuição desse trabalho é a proposta e implementação de um controlador de configurações implementado em hardware. Sendo de uso geral para escalonamento estático de reconfigurações, RSCM pode ser utilizado por sistemas reconfiguráveis com uma ampla faixa de características desde que empreguem uma política de escalonamento estático. A simplificação ou reformulação do sistema RSCM básico para ambientes específicos é simples e foi apresentado um estudo de caso de tal adaptação para o projeto R8R, descrito na Seção 6.2.

O trabalho realizado contribui também de forma estratégica para o desenvolvimento e a implementação de SDRs. Através dele, pôde-se melhor entender o processo de configuração parcial de dispositivos reconfiguráveis.

Num primeiro instante, apresentou-se conceitos relativos à reconfiguração dinâmica e parcial. A seguir, a proposta de um modelo genérico de sistema reconfigurável, o modelo GRS, foi apresentada na Seção 1.2.1. Ainda no mesmo Capítulo, um fluxo de desenvolvimento e também um conjunto de ferramentas necessárias no projeto SDRs foi apresentado, na Seção 1.4. PaDReH é uma proposta de arcabouço para projeto e implementação de SDRs. Para que o leitor pudesse entender melhor o processo de configuração, uma proposta de critérios para classificar os processos de reconfiguração foi apresentada na Seção 2.1.

Os modelos e implementações de controladores de configurações forneceram o embasamento necessário para melhor entender a tarefa de controle de configurações em SDRs. Além disso, um comparativo entre esses modelos e implementações foi realizado, como apresentado na Seção **??**. Isto também possibilitou a proposta de um modelo de estrutura geral para controladores de configurações, apresentado na Figura 3.7.

7.2 Conclusões

Neste trabalho, pôde-se compreender o cenário atual do desenvolvimento de sistemas dinâmica e parcialmente reconfiguráveis. Ficou evidenciada a falta de ferramentas para o projeto e suporte à implementação de sistemas desta natureza. Embora diversos estudos sejam realizados enfatizando RTR, os fabricantes ainda não disponibilizam dispositivos que habilitem a sua reconfiguração dinâmica e parcial de uma forma mais facilmente integrável ao fluxo de projeto tradicional de sistemas computacionais. O fluxo de projeto convencional não se adequa corretamente ao projeto de SDRs e, na grande maioria dos casos é necessário utilizar estratégias alternativas para desenvolver SDRs, tal como o fluxo de projeto modular, proposto pela Xilinx [BRI03].

A necessidade de um subsistema para suporte a operação de SDRs é evidente. RSCM é uma proposta que visa suprir a necessidade de um controlador de configurações implementado em hardware, ocasionando baixa sobrecarga de área do dispositivo reconfigurável. Propostas anteriores como [CUR03] e [BLO03], são de uso mais específico que o RSCM. Um deles, [BLO03], é específico para configurar os transceptores de dispositivos VirtexII e o outro aplica apenas reconfigurações incrementais. O RSCM é utilizado para controlar a reconfiguração de núcleos IP arbitrariamente complexos.

RSCM teve sua implementação em hardware como objetivo inicial de investigar tal abordagem. Implementações anteriores de controladores de configurações foram em software. Esta investigação poderia apresentar melhores resultados se as outras implementações fossem melhores documentadas ou ainda se pudessem ser acessadas sem maiores restrições, até mesmo para a realização de simulações. Como os autores não apresentam resultados sobre suas implementações, tais como tempos de reconfiguração e área ocupada, não foi possível elaborar um comparativo entre as implementações em software e hardware. Dessa forma, é difícil estimar qual seria a melhor estratégia de implementação de um controlador de configurações.

Foi possível implementar um sistema de controle de configurações de forma completa, com a possibilidade de ser parametrizado de acordo com o SDR a ser implementado. O controle de configurações é um processo complexo que requer o gerenciamento de diversos fatores, tais como o estado de operação do SDR, o uso de escalonamento de bitstreams pré-definido, o armazenamento de bitstreams parciais e também o controle do processo de configuração.

Ainda não foi possível abordar quantitativamente o desempenho do controlador RSCM, como seria necessário. O objetivo primordial deste trabalho foi prover um controlador reutilizável e operacional. Acredita-se que tal objetivo tenha sido plenamente atingido.

7.3 Trabalhos futuros

Outros estudos ainda devem ser realizados para melhor entender como desenvolver e implementar SDRs, de maneira a difundir esta estratégia de implementação. O controle de configurações ainda carece de maior capacidade de controle, sobretudo no que diz respeito a implicações relacionadas a sua implementação em hardware ou software. Por isso, uma tarefa importante a ser realizada é procurar maiores detalhes sobre as implementações em software, ou até mesmo desenvolver um controlador de configurações em software. Esta última alternativa é uma opção vista como extremamente interessante.

O desenvolvimento de outros SDRs, como estudo de caso, também constitui um possível trabalho futuro. Até mesmo prover variações do processador R8R reconfigurável poderá representar um avanço significativo nas pesquisas da área que contribui para a certificação da importância de empregar técnicas RTR no desenvolvimento e implementação de sistemas de hardware. Uma possível variação do R8R deve lidar com mais de uma área reconfigurável. Para isso, o CCC personalizado para o R8R com uma área deve ser ligeiramente modificado. Outro caminho é trabalhar com instruções não bloqueantes na R8R, para aumentar o desempenho de software que pode iniciar a configuração de coprocessadores antes de serem necessários para amortizar o tempo de reconfiguração.

Melhorar o desempenho do controlador RSCM é uma tarefa importante, que implica em maiores estudos sobre a interface ICAP, a fim de torná-la funcional e, adotar a alternativa mais eficiente de memórias de configurações. Os módulos desenvolvidos também podem ser modificados de maneira a possuírem um melhor desempenho e também representarem uma menor sobrecarga de área.

A relocação de bitstreams é outro tópico importante a ser tratado. Um módulo relocador deve ser anexado ao controlador RSCM, de maneira a reduzir a quantia de memória necessária para armazenamento de configurações. O estudo de técnicas de relocação em dispositivos VirtexII pode contribuir para obtenção de informações sobre a arquitetura de configuração dos dispositivos desta família, informações estas muito escassas. Em [STE03], Platzner et al. dizem ter solucionado o problema de relocação de bitstreams para dispositivos da série Virtex da Xilinx, mas maiores detalhes não são fornecidos.

A maioria das estruturas que armazenam informações internas no RSCM são implementadas em hardware, tais como a Tabela de Dependência e Descritores e a Tabela de Alocação de Recursos. Isso implica em uma tarefa de ressíntese do sistema toda vez que estas estruturas devem ser modificadas. Uma outra tarefa importante é prover meios mais práticos de parametrizar o controlador de configurações, tornando-o mais flexível e, portanto mais reutilizável.

A dificuldade em verificar o correto funcionamento de SDRs aponta para o estudo de técnicas para simular o comportamento de sistemas desta natureza. Até então, o sistema

deve ser prototipado para que se possa verificar seu funcionamento.

A descrição de SDRs em níveis mais altos que RTL sugere o estudo de técnicas para utilizar outras linguagem de descrição, tal como SystemC. Modelar um sistema para controle de configurações em alto nível de abstração constitui uma alternativa interessante como trabalho futuro.

Referências Bibliográficas

[ALG89] Algotronix, Ltd. **Configurable Array Logic 1024**. Data Sheet, Abril 1989. 41p.

[ALT03] Altera, Corp. **Stratix GX FPGA Family (V1.2)**. Data Sheet, Março 2003. 216p.

[ATM00] Atmel, Corp. **At40k Series Configuration**. Application Note, Janeiro 2000. 41p.

[ATM00a] Atmel, Corp. **Implementing Cache Logic with FPGAS**. Application Note, Setembro 2000. 5p.

[ATM00b] Atmel, Corp. **AT6000(LV) Series**. Application Note, Setembro 2000. 28p.

[BER00] Bergamaschi, R.A.; Lee, W.R. **Designing systems-on-chip using cores**. In: 37^{th} Design Automation Conference (DAC'00), Los Angeles, CA, USA. Junho 2000. pp.420-425.

[BLO03] Blodget, B.; McMillan, S.; Lysaght, P. **A Lightweight Approach for Embedded Reconfiguration of FPGAs**. In: 6^{th} Design Automation and Test in Europe Conference and Exhibition (DATE'03), Messe Munich, Germany. Março 2003. pp.399-400.

[BRE01] Brebner, G.; Diessel, O. **Chip-Based Reconfigurable Task Management**. In: 11^{th} Field-Programmable Logic and Applications (FPL'01), Belfast, Northern Ireland, UK. Agosto 2001. pp.182-191.

[BRI02] Brião, E.W. **Configuração, Reconfiguração Total e Reconfiguração Parcial de Hardware sobre a plataforma Memec-Insight V2MB1000.** Trabalho Individual II. Pontifícia Universidade Católica do Rio Grande do Sul - PPGCC - FACIN. Porto Alegre, RS, Brasil. Novembro 2002. 54p.

[BRI03] Brião, E.W. **Reconfiguração Parcial e Dinâmica para Núcleos de Propriedade Intelectual com Interfaces de Comunicação Padronizadas**. Seminário de Andamento. Pontifícia Universidade Católica do Rio Grande do Sul - PPGCC - FACIN. Porto Alegre, RS, Brasil. Julho 2003. 17p.

[BRO92] Brown, S.D.; Francis, R.J.; Rose, J.; Vranesic, Z. **Field-Programmable Gate Arrays**. Chapter 1 - Introduction to FPGAs. Kluwer Academic Publishers, Maio 1992. pp.1-11.

[BUR97] Burns, J.; Donlin, A.; Hogg, J.; Singh, S.; Wit, M. de. **A Dynamic Reconfiguration Run-Time System**. In: 5^{th} Field-Programmable Custom Computing Machines (FCCM'97), Napa Valley, CA, USA. Abril 1997. pp.66-75.

[CAL01] Calazans, N.L.V.; Moraes, F.G. **Integrating the Teaching of Computer Organization and Architecture with Digital Hardware Design Early in Undergraduate Courses**. IEEE Transactions on Education, vol.44-2, Maio 2001. pp.109-119.

[CHO99] Chow, P.; Rose, J.; Seo, S.; Chung, K.; Rahardja, I.; Paez, G. **The Design of an SRAM-Based Field-Programmable Gate Array: Part I: Architecture**. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.7-2, Junho 1999. pp.191-197.

[CHO99a] Chow, P.; Rose, J.; Seo, S.; Chung, K.; Rahardja, I.; Paez, G. **The Design of an SRAM-Based Field-Programmable Gate Array: Part II: Circuit Design and Layout**. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.7-3, Setembro 1999. pp.321-330.

[COM00] Compton, K.; Cooley, J.; Knol, S.; Hauck, S. **Configuration Relocation and Defragmentation for Reconfigurable Computing**. In: 8^{th} Field-Programmable Custom Computing Machines (FCCM'00), Napa Valley, CA, USA. Abril 2000. pp.279-280.

[COM02] Compton, K.; Hauck, S. **Reconfigurable Computing: a survey of systems and software**. ACM Computing Surveys, vol.34-2, Junho 2002. pp.171-210.

[COM02a] Compton, K.; Li, Z.; Cooley, J.;Knol, S.; Hauck, S. **Configuration Relocation and Defragmentation for Run-Time Reconfigurable Computing**. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.10-3, Junho 2002. pp.209-220.

[CUR03] Curd, D.R. **Partial Reconfiguration of RocketIO Pre-emphasis and Differential Swing Control Attributes**. Xilinx Application Note - XAPP660 (v1.0), Janeiro 2003. 13p.

[DEH98] DeHon, A. **Comparing Computing Machines**. In: Configurable Computing: Technology and Applications, (SPIE 3526), Boston, MA, USA. Novembro 1998. pp.124-133.

[DEH00] DeHon, A. **The Density Advantage of Configurable Computing**. IEEE Computer, vol.33-4, Abril 2000. pp.41-49.

[DYE02] Dyer, M.; Plessl, C.; Platzner, M. **Partially Reconfigurable Cores for Xilinx Virtex**. In: 12^{th} Field Programmable Logic and Application (FPL'02), La Grande-Motte, Montpellier, France. Setembro 2002. pp.292-301.

[GUC99] Guccione, S.A.; Levi, D.; Sundararajan, P. **JBits: Java based interface for reconfigurable computing**. In: 2^{nd} Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD'99). Laurel, MA, USA. Setembro 1999. 9p.

[GUP97] Gupta, R.K.; Zorian, Y. **Introducing Core-Based System Design**. IEEE Design and Test of Computers, vol.14-4, Outubro-Dezembro 1997. pp.15-25.

[HAU98] Hauck, S. **The Future of Reconfigurable Systems**. In: 5^{th} Canadian Conference on Field Programmable Devices (FPD'98), Montreal, Canada. Junho 1998. 7p.

[HOR01] Horta, E.L.; Lockwood, J.W. **PARBIT: A Tool to Transform Bitfiles to Implement Partial Reconfiguration of Field Programmable Gate Arrays (FPGAs)**. Technical Report (WUCS-01-13). Washington University in Saint Louis, Department of Computer Science. Julho 2001. 37p.

[HOR02] Horta, E.L.; Lockwood, J.W.; Taylor, D.E.; Parlour, D. **Dynamic Hardware Plugins in an FPGA with Partial Run-time Reconfiguration**. In: 39^{th} Design Automation Conference (DAC'02), New Orleans, LA, USA. Junho 2002. pp.343-348.

[HOR02a] Horta, E.L.; Lockwood, J.W.; Kofuji, S.T. **Using PARBIT to Implement Partial Run-Time Reconfigurable Systems**. In: 12^{th} Field Programmable Logic and Application (FPL'02), La Grande-Motte, Montpellier, France. Setembro 2002. pp.182-191.

[IBM99] IBM, Corp. **The CoreConnect Bus Architecture**. IBM Microelectronics Data Sheet, Março 1999. 8p.

[LIM02] Lim, D.; Peattie, M. **Two Flows for Partial Reconfiguration: Module Based or Small Bit Manipulations**. Xilinx Application Note - XAPP290 (v1.0), Maio 2002. 23p.

[LYS97] Lysaght, P. **Towards an Expert System for a *priori* Estimation of Reconfiguration Latency in Dynamically Reconfigurable Logic**. In: 7^{th} Field-Programmable Logic and Applications (FPL'97), London, UK. Setembro 1997. pp.183-192.

[MAE01] Maestre, R.; Kurdahi, F.J.; Fernández, M.;Hermida, R.; Bagherzadeh, N.; Singh, H. **A Formal Approach to Context Scheduling for Multicontext Reconfigurable Architectures**. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.9-1, Fevereiro 2001. pp.173-185.

[MAE01a] Maestre, R.; Kurdahi, F.J.; Fernández, M.; Hermida, R.; Bagherzadeh, N.; Singh, H. **A Framework for Reconfigurable Computing: Task Scheduling and Context Management**. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.9-6, Dezembro 2001. pp.858-873.

[MAR01] Martin, G.; Chang, H. **Tutorial - System on Chip Design**. In: 9^{th} International Symposium on Integrated Circuits, Devices & Systems (ISIC'01), Marina Mandarin, Singapore. 2001. pp.12-17.

[MAR02] Marcon, C.A. **Análise do Particionamento e Escalonamento de Recursos para o Projeto Integrado de Sistemas de Hardware/Software**, Exame de Qualificação. Pontifícia Universidade Católica do Rio Grande do Sul - PPGCC - FACIN. Porto Alegre, RS, Brasil. Junho 2002. 62p.

[MCG99] McGregor, G.; Lysaght, P. **Self Controlling Dynamic Reconfiguration: A Case Study**. In: 9^{th} Field-Programmable Logic and Applications (FPL'99), Glasgow, UK. Agosto-Setembro 1999. pp.144-154.

[MCM00] McMillan, S.; Guccione, S.A. **Partial Run-Time Reconfiguration using JRTR**. In: 10^{th} Field-Programmable Logic and Applications (FPL'00), Villach, Austria. Agosto 2000. pp.352-360.

[MEM02] Memec Insight. **Virtex-II^{TM} V2MB1000 Development Board**. User's Guide (V1.3), Fevereiro 2002. 43p.

[MEM02a] Memec Insight. **P160 Protitype Module**. User's Guide (V1.1), Fevereiro 2002. 14p.

[MER98] Merino, P.; Jacome, M.; López, J.C. **A Metodology for Task Based Partitioning and Scheduling of Dynamic Reconfigurable Systems**. In: 6^{th} Field-Programmable Custom Computing Machines (FCCM'98), Napa Valley, CA, USA. Abril 1998. pp.324-325.

[MES02] Mesquita, D. **Contribuições para Reconfiguração Parcial, Remota e Dinâmica de FPGAs**. Dissertação de Mestrado. Pontifícia Universidade Católica do Rio Grande do Sul - PPGCC - FACIN. Porto Alegre, RS, Brasil. Março 2002. 99p.

[MIC03] Micron Technology, Inc. **Double Data Rate (DDR) SDRAM**. Data Sheet (rev03/03), Março 2003. 81p.

[MöL03] Möller, L.H. **Ferramentas de Reconfiguração Parcial, Remota e Dinâmica de FPGAs Virtex**. Relatório Técnico (RT035). Pontifícia Universidade Católica do Rio Grande do Sul - PPGCC - FACIN. Porto Alegre, RS, Brasil. Dezembro 2003. 29p.

[MOR03] Moreno, E.I. **Modelando, descrevendo e validando NoCs para SoCs em diferentes níveis de abstração**. Seminário de Andamento. Pontifícia Universidade Católica do Rio Grande do Sul - PPGCC - FACIN. Porto Alegre, RS, Brasil. Julho 2003. 14p.

[MOR03a] Moraes, F.G.; Calazans, N.L.V. **R8 Processor Architecture and Organization Specification and Design Guidelines**, Capturada em: http://www.inf.pucrs.br/~gaph/Projects/R8/public/R8_arq_spec_eng.pdf, Julho 2003. 18p.

[NAT98] National Semiconductor, Corp. **National Complementary Gallium Arsenide Configurable Logic Array**. Navigator Magazine, vol.14, Julho 1998. 1p.

[OST03] Ost, L.C. **Redes Intra-Chip Parametrizáveis com Interface Padrão OCP para Síntese em Hardware**. Seminário de Andamento. Pontifícia Universidade Católica do Rio Grande do Sul - PPGCC - FACIN. Porto Alegre, RS, Brasil. Julho 2003. 15p.

[PAL01] Palma, J.C.S. **Métodos para desenvolvimento e distribuição de IP-cores**. Dissertação de Mestrado. Pontifícia Universidade Católica do Rio Grande do Sul - PPGCC - FACIN. Porto Alegre, RS, Brasil. Dezembro 2001. 92p.

[PAL02] Palma, J.C.S.; Mello, A.V.; Möller, L.H.; Moraes, F.G.; Calazans, N.L.V. **Core Communication Interface for FPGAs**. In: 15^{th} Symposium on Integrated Circuits and Systems Design (SBCCI'02), Porto Alegre, RS, Brazil. Setembro 2002. pp.183-188.

[POR02] Porrmann, M.; Witkowski, U.; Kalte, H.; Rückert, U. **Dynamically Reconfigurable Hardware - A New Perspective for Neural Network Implementations**. In: 12^{th} Field Programmable Logic and Application (FPL'02), La Grande-Motte, Montpellier, France. Setembro 2002. pp.1048-1057.

[RAG02] Raghavan, A.; Sutton, P. **JPG - A Partial Bitstream Generation Tool to Support Partial Reconfiguration in Virtex FPGAs**. In: 16^{th} International Parallel and Distributed Processing Symposium (IPDPS'02), Fort Lauderdale, Florida, USA. Abril 2002, pp.155-160.

[ROB99] Robinson, D.; Lysaght, P. **Modelling and Synthesis of Configuration Controllers for Dynamically Reconfigurable Logic Systems using the DCS CAD Framework**. In: 9^{th} Field-Programmable Logic and Applications (FPL'99), Glasgow, UK. Agosto-Setembro 1999. pp.41-50.

[SAN99] Sanchez, E.; Sipper, M.; Haenni, J-O.; Beuchat, J-L.; Stauffer, A.; Perez-Uribe, A. **Static and dynamic configurable systems**. IEEE Transactions on Computers, vol.48-6, Junho 1999. pp.556-564.

[SAR01] Sarmento, M. **Arquiteturas Auto-Reconfiguráveis em Sistemas Digitais**. Trabalho de Conclusão II. Pontifícia Universidade Católica do Rio Grande do Sul - FACIN. Porto Alegre, RS, Brasil. Junho 2001. 101p.

[SHI98] Shirazi, N.; Luk, W.; Cheung, P.Y.K. **Run-Time Management of Dynamically Reconfigurable Designs**. In: 8^{th} Field-Programmable Logic and Applications (FPL'98), Tallinn, Estonia. Agosto-Setembro 1998. pp.59-68.

[SMI02] Smit, G.J.M.; Havinga, P.J.M.; Smit,L.T.; Heysters, P.M.; Rosien, M.A.J. **Dynamic Reconfiguration in Mobile Systems**. In: 12^{th} Field Programmable Logic and Application (FPL'02), La Grande-Motte, Montpellier, France. Setembro 2002. pp.171-181.

[STE03] Steiger, C.; Walder, H.; Platzner, M. **Heuristics for Online Scheduling Real-time Tasks to Partially Reconfigurable Devices**. In: 13^{th} International Conference on Field Programmable Logic and Application (FPL'03), Lisbon, Portugal. Setembro 2003. pp.575-584.

[TAN01] Tanenbaum, A.S. **Modern Operating Systems**. Upper Saddle River, NJ Prentice-Hall PTR, 2001. 728p.

[TAY01] Taylor, D.E.; Turner, J.S.; Lockwood, J.W. **Dynamic Hardware Plugins (DHP): Exploiting Reconfigurable Hardware for High-Performance Programmable Routers**. In: 4^{th} Open Architectures and Network Programming (OPENARCH'01), Anchorage, Alaska, USA. Abril 2001. pp.25-35.

[TAY02] Taylor, D.E.; Turner, J.S.; Lockwood, J.W.; Horta, E.L. **Dynamic Hardware Plugins (DHP): Exploiting Reconfigurable Hardware for High-Performance Programmable Routers**. Computer Networks, vol.38-3, Fevereiro 2002. pp.295-310.

[TOS01] Toshiba, Corp. **TH50VSF2580/2581AASB. Toshiba Multi-Chip Integrated Circuit Silicon Gate CMOS**. Data Sheet, Março 2001. 50p.

[VAH03] Vahid, F. **The Softening of Hardware**. IEEE Computer, vol.36-4, Abril 2003. pp.27-34.

[VAS95] Vasilko, M.; Ait-Boudaoud, D. **Scheduling for Dynamically Reconfigurable FPGAs**. In: 7^{th} International Workshop on Logic and Architecture Synthesis (IFIP TC10 WG10.5), Grenoble, France. Dezembro 1995. pp.328-336.

[VIL97] Villasenor, J.; Mangione-Smith, W.H. **Configurable Computing**. Scientific American, vol.276-6, Junho 1997. pp.54-59.

[VIS03] Vissers, K.A. **Parallel Processing Architectures for Reconfigurarable System**. In: 6^{th} Design Automation and Test in Europe Conference and Exhibition (DATE'03), Messe Munich, Germany. Março 2003. pp.396-397.

[WAL02] Waller, L. **The Big Question in Counting FPGA Gates: should memory be included?** Capturada em: http://www.eedesign.com/editorial/1997/fpgacolumn9710.html, Agosto de 2002. 5p.

[WAL03] Walder, H.; Platzner, M. **Online Scheduling for Block-partitioned Reconfigurable Devices**. In: 6^{th} Design Automation and Test in Europe Conference and Exhibition (DATE'03), Messe Munich, Germany. Março 2003. pp.290-295.

[WIR95] Wirthlin, M.J.; Hutchings, B.L. **A Dynamic Instruction Set Computer**. In: 3^{rd} Field-Programmable Custom Computing Machines (FCCM'95), Los Alamitos, CA, USA. Abril 1995. pp.99-107.

[WIR95a] Wirthlin, M.J.; Hutchings, B.L. **DISC: The Dynamic Instruction Set Computer**. In: Field Programmable Gate Arrays (FPGAs) for Fast Board Development and Reconfigurable Computing (SPIE 2607), Philadelphia, PA, USA. Outubro 1995. pp.92-103.

[WIR97] Wirthlin, M.J.; Hutchings, B.L. **Improving Functional Density Using Run-Time Circuit Reconfiguration**. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.6-2, Junho 1997. pp.247-256.

[XIL97] Xilinx, Inc. **XC6200 Field Programmable Gate Arrays**. Product Description (v1.1), Abril 1997. 73p.

[XIL00] Xilinx, Inc. **Virtex Series Configuration Architecture User Guide**. Xilinx Application Note - XAPP151 (v1.5), Setembro 2000. 45p.

[XIL01] Xilinx, Inc. **Virtex^{TM} 2.5V: Field-Programmable Gate Array**. Product Specification (v2.5), Abril 2001. 74p.

[XIL02] Xilinx Inc. **MicroBlaze^{TM} Hardware**. Reference Guide, Março 2002. 222p.

[XIL02a] Xilinx Inc. **MicroBlaze^{TM} Software**. Reference Guide, Abril 2002. 198p.

[XIL02b] Xilinx, Inc. **Virtex-II 1.5V: Field-Programmable Gate Array**. Advance Product Specification (v1.9), Setembro 2002. 309p.

[XIL02c] Xilinx, Inc. **Virtex-II Platform FPGA**. User's Guide (v1.5), Dezembro 2002. 460p.

[XIL02d] Xilinx, Inc. **Virtex-II Series Configuration Architecture**. User's Guide, Maio 2002. 10p. (*Private Communication*).

[XIL03] Xilinx, Inc. **Virtex-II Pro^{TM} Platform FPGAs: Complete Data Sheet**. Advance Product Specification, Setembro 2003. 407p.

[XIL03a] Xilinx, Inc. **Embedded System Tools Guide - Embedded Development Kit EDK 6.1**. User's Guide (v1.0), Outubro 2003. 386p.

[ZHA00] Zhang, X.; Ng, K.W. **A review of high-level synthesis for dynamically reconfigurable FPGAs**. Microprocessors and Microsystems, vol.24, Abril 2000. pp.199-211.

Apêndice A

Programas utilizados no projeto R8R

Neste Apêndice, apresenta-se os códigos dos programas utilizados para validar o processador reconfigurável R8R. os mesmo também foram utilizados para obter as estimativas sobre a validade do emprego da resolução de problemas computacionais através do emprego de coprocessadores implementados em hardware, em contra proposta aos mesmos implementados através de rotinas em software.

A.1 Programa de teste do processador R8R

A listagem a seguir apresenta o código da aplicação desenvolvida para testar o funcionamento do sistema R8R. Todos os coprocessadores desenvolvidos são testados e, também, a interface com o Controlador RSCM é validada.

```
1  .CODE
2     xor r0,r0,r0
3     ldl r1,#01h
4     ldh r1,#00h
5     ldl r2,#86h
6     ldh r2,#00h
7     ldl r3,#02h
8     ldh r3,#00h
9     ldl r4,#AAh
10    ldh r4,#00h
11    ldl r5,#03h
12    ldh r5,#00h
13    ldl r6,#F7h
14    ldh r6,#00h
15    ldl r7,#04h
16    ldh r7,#00h
17    ldl r8,#0Bh
18    ldh r8,#01h
19
20    ldl r15,#4avg
21    ldh r15,#4avg
22
23    initr #1
24    wrr r1,r2
25    wrr r3,r4
26    wrr r5,r6
27    wrr r7,r8
```

```
28      rdr r11,r12
29      st r15,r12,r0
30
31      ldl r15,#2avg
32      ldh r15,#2avg
33      selr #2
34      initr #2
35      wrr r1,r2
36      wrr r3,r4
37      rdr r11,r12
38      st r15,r12,r0
39
40      ldl r1,#FEh
41      ldh r1,#00h
42
43      ldl r2,#0Fh
44      ldh r2,#00h
45
46      ldl r3,#E4h
47      ldh r3,#0Eh
48
49      ldl r4,#84h
50      ldh r4,#03h
51
52      ldl r10,#produto
53      ldh r10,#produto
54
55      ldl r11,#divisao
56      ldh r11,#divisao
57
58      ldl r12,#resto
59      ldh r12,#resto
60
61      ldl r13,#raizq
62      ldh r13,#raizq
63
64      ldl r15,#01h
65      ldh r15,#00h
66
67      selr #3
68      initr #3
69      wrr r1,r2
70      rdr r5,r6       ; r6 é o produto
71
72      st r6,r10,r0    ;pmem(r10+r0) <- r6
73
74      selr #4
75      initr #4
76      wrr r3,r2       ; EE4h div FEh
77      rdr r5,r6       ; r5 = resto  r6 = quociente
78
79      st r5, r12, r0
80      st r6, r11, r0
81
82      selr #5
83      initr #5
84      wrr r15,r4
85      rdr r5,r6
86
87      st r6,r13,r0
88
89      halt
90
91  .ORG #024H
92
```

```
93  .DATA
94     avg4:    DB #0000H
95     avg2:    DB #0000H
96     produto: DB #0000H
97     divisao: DB #0000H
98     resto:   DB #0000H
99     raizq:   DB #0000H
100 .ENDDATA
```

A.2 Multiplicação implementada em software

A listagem a seguir apresenta o código da rotina desenvolvida para realizar tarefa similar a realizada pelo coprocessador ***multi***.

```
1   .CODE
2      xor r14,r14,r14
3      ldl r1,#02h    ; load multiplicand into r1
4      ldh r1,#00h
5
6      ldl r2,#02h    ; load multiplier into r2
7      ldh r2,#00h
8
9      ldl r15,#01h   ; load constant "1" into r15
10     ldh r15,#00h   ;
11
12     ldl r3,#00h
13     ldh r3,#00h
14
15  loop:
16     and r0,r2,r15   ; r0 <- r2 and r15 (1)
17     or r7,r2,r14    ; if r2 = 0?:
18     jmpzd #finish   ; then go to #finish
19     or r7,r0,r14    ; r0 <- r0 or r14 (0)
20     jmpzd #loopend  ; if r0 = 0 then go to #loopend
21     add r6,r3,r14   ; r6 <- r3
22     add r6,r1,r6    ; r6 <- r1 + r6
23     add r3,r6,r14   ; r3 <- r6
24
25  loopend:
26     sr0 r2,r2       ;  r2 << 1
27     sl0 r1,r1       ;  r1 >> 1
28     jmpd #loop
29
30  finish:
31     halt
32
33  .ORG #015H
34  .DATA
35     product:  DB #0000H
36  .ENDDATA
```

A.3 Divisão implementada em software

A listagem a seguir apresenta o código da rotina desenvolvida para realizar tarefa similar a realizada pelo coprocessador ***div***.

```
1   .CODE
2      xor R0,R0,R0         ; R0 <- 0
```

```
3     ldl R1,#8Fh          ; Quociente <- 9
4     ldl R2,#0Fh          ; Div <- 4
5     ldl R4,#0Fh          ; R4 <- 15
6     ldh R7,#80h          ; R7 <- 1000000
7
8     and R6,R7,R1         ; R6 <- (R1 and R7)
9     jsrd #flag           ; Vai para subrotina "flag"
10    sl0 R1,R1            ; Quociente <- SHL(Resto) &0
11    jsrd #carry          ; Vai para subrotina "carry"
12
13
14  loop: sub R3,R3,R2     ; Resto <- Resto - Div
15    jmpnd #neg           ; if (negativo) goto "neg"
16
17
18  pos:  and R6, R7, R1   ; R6 <- (R1 and R7)
19    jsrd #flag           ; Vai para subrotina "flag"
20    sl1 R1,R1            ; SHL(quociente) &1
21    jsrd #carry          ; Vai para subrotina "carry"
22    jmpd #testafim       ; Vai para "testafim"
23
24
25  neg:  add R3,R3,R2     ; Resto <- Resto + Div
26    and R6,R7,R1         ; R6 <- (R1 and R7)
27    jsrd #flag           ; Vai para subrotina "flag"
28    sl0 R1, R1           ; SHL(Quociente) &0
29    jsrd #carry          ; Vai para subrotina "carry"
30    jmpd #testafim       ; Vai para "testafim"
31
32  carry:  subi R5,#01h   ; flag <- flag - 1
33    jmpzd #C             ; if (zero) goto C
34
35                         ; (else)
36  noC:  sl0 R3, R3       ; Resto <- ssl(Resto) &0
37    rts                  ; retorna
38
39  C: sl1 R3, R3          ; Resto <- ssl(Resto) &1
40    rts                  ; retorna
41
42  flag: jmpnd #setflag   ; se R6 = negativo, goto "setflag"
43    xor R5, R5, R5       ; R5 <- 0
44    rts                  ; retorna
45
46  setflag:xor R5,R5,R5   ; R5 <- 0
47    ldl R5,#01h          ; flag <- 1
48    rts                  ; retorna
49
50  testafim:subi R4, #01  ; R4 <- R4 - 1
51     JMPND #fim          ; if (negativo) goto "fim"
52     JMPD #loop          ; else go to "loop"
53
54  fim:  sr0 R3, R3
55    halt
56
57  .ENDCODE
```

A.4 Raiz quadrada implementada em software

A listagem a seguir apresenta o código da rotina desenvolvida para realizar tarefa similar a realizada pelo coprocessador ***sqrt***.

```
1   .CODE
2      xor r0,r0,r0
3      ldl r1,#FFh
4      ldh r1,#0Fh
5
6      addi r1,#01h
7      sr0 r1,r1
8
9      ldl r2,#FFh
10     ldh r2,#FFh
11
12  ciclo1:
13     addi r2,#01h
14     sub r1,r1,r2
15     jmpnd #fim1
16     jmpzd #fim1
17     jmpd #ciclo1
18
19  fim1:
20
21     ldl r7,#c1
22     ldh r7,#c1
23     st r2,r7,r0    ; pmem(r7+r0) <- r2 (dado do cop)
24     halt
25
26  .ORG #10H
27  .DATA
28     C1:  DB #FFFFH
29  .ENDDATA
```