**Pontifícia Universidade Católica do Rio Grande do Sul**

**Faculdade de Informática**

**Pós-Graduação em Ciência da Computação**

# Infra-estrutura e Implementação de Controle de Configurações em Software para Hardware Reconfigurável

Rafael Iankowski Soares

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

Orientador: Prof. Dr. Ney Laert Vilar Calazans

Porto Alegre, maio de 2006.

*Catalogação.*

**Agradecimentos**

Agradeço, primeiramente, aos meus familiares e principalmente aos meus pais pelo grande incentivo, conselhos e palavras que me confortaram nos dias difíceis desta caminhada.

Agradeço ao meu orientador, professor Ney Calazans, pela excelente receptividade neste programa, pois quando eu cheguei estava totalmente perdido em meio a tantas novidades. Agradeço também sua paciência e devoção ao trabalho que fizeram com que eu conseguisse concluí-lo com êxito.

Um agradecimento à CAPES por financiar meu trabalho nesta instituíção, sem este não seria possível minha estada em Porto Alegre.

Agradeço aos meus amigos que proporcionaram momentos de alegria e descontração em meio a tantas dificuldades, sem eles tudo seria mais difícil. Agradeço aos meus grandes "broteis"(Márcio e Odorico) que dividiram o apto nesta etapa, foi muito legal a convivência, as festas e churrascos que realizamos durante estes 2 anos.

Agradeço aos membros do GAPH que me ajudaram no trabalho resolvendo vários problemas, a ajuda de vocês foi fundamental. Aos mestrandos que compartilharam vários momentos difíceis, a vocês um muito obrigado. Um agradecimento especial ao Molar, você foi um "Co-Orientador"pra mim, devo grande parte da realização de meu trabalho a você, muito obrigado por tudo! Também um agradecimento especial ao Carvalho (Emerson) tão citado em minha dissertação, você foi peça importante neste trabalho, valeu por todos os conselhos.

*"Dar menos que seu melhor é sacrificar o dom que você recebeu."* *(Steve Prefontaine)*

## Resumo

A evolução de sistemas integrados em um único chip (*SoCs*) tem exigido o desenvolvimento de técnicas de projeto que superem gargalos surgidos da mera adaptação de técnicas tradicionais de projetos de circuitos digitais a tais sistemas. Ao lado de integração de multiprocessadores homogêneos, ocorrendo na área de processadores para PCs, e múltiplos processadores distintos em um único substrato de silício (*MPSoCs*), encontra-se o uso de *hardware reconfigurável*. Estes são sistemas digitais onde a flexibilidade de personalização pode atingir níveis maiores do que em um processador que executa software. Enquanto a flexibilidade de um processador vem da capacidade de executar diferentes seqüências de instruções de uma memória externa, a flexibilidade da reconfigurabilidade deriva da chance de alterar interconexão e funcionalidade de componentes que definem a estrutura do sistema digital. Assim, trata-se de flexibilizar a estrutura do hardware de forma similar à provida no seqüenciamento de instruções por um programador. A esta flexibilidade corresponde uma complexidade de projeto e controle elevada. Ademais, o avanço de infra-estruturas de suporte ao controle de sistemas reconfiguráveis é incipiente quando comparado com tarefas equivalentes em software. Visando contribuir para superar as limitações que impedem o uso amplo de hardware reconfigurável, o presente trabalho propõe uma arquitetura alvo para sistemas dinâmica e parcialmente reconfiguráveis (SDRs) auto-reconfiguráveis. Esta arquitetura alvo pressupõe a existência de um processador programável na parte não-reconfigurável do sistema. Assume-se ainda que este processador pode dedicar parte de seu tempo para controlar o processo de controle de configuração do SDR. Adicionalmente, propõe-se, valida-se e prototipa-se uma infraestrura que habilita a implementação de leiautes válidos para SDRs autoreconfiguráveis. Também sugere-se um fluxo de projeto utilizável para implementar de forma semi-automatizada SDRs que usem a arquitetura alvo e a infraestrutura de projeto propostas. Sobre o processador alvo, propõe-se a implementação de um software controlador de configurações baseado em modelo já proposto e validado em hardware (RSCM). O novo controlador, RSCM-S, foi prototipado em FPGAs Xilinx, e disponibiliza-se dados preliminares de comparação entre RSCM e RSCM-S. Estudos de caso demonstram a viabilidade dos conceitos propostos, incluindo uma aplicação reconfigurável de criptografia.

**Abstract**

The evolution of Systems on chip (*SoCs*) requires the development of design techniques to overcome bottlenecks derived from the adaptation of traditional digital circuits design techniques to such systems. Besides integrating multiple homogeneous processors, as occurs in the PC processor arena, and techniques for providing multiple heterogeneous processor in a single silicon substrate (*MPSoCs*), there is the use of *reconfigurable hardware* (RH). These are digital systems where behavior customization can attain a higher degree than what is achievable with software executing processors. While the flexibility of a processor derives from the capacity to execute different sequences of instructions stored in memory, the flexibility of RH derives from the possibility of altering the interconnection and the functionality of components which define the system structure in itself. Thus, RH makes the hardware structure flexible in the same way the use of RAM memories for the program memory of processors make the processor able to execute virtually any task. However, to this enhanced flexibility corresponds increased complexity for design and control, when compared to software. The proposition of support and control of reconfigurable systems is largely behind what has already been achieved for software systems. Contributing to overcome the limitations that prevent RH use to become widespread, this work proposes a target architecture for partially and dynamically reconfigurable systems (DRSs) that are also self-reconfigurable. The architecture assumes the existence of a processor in the non-reconfigurable part of the system. It also assumes that this processor can encompass the processes involved in the DRS configuration control. Besides, the work proposes, validates and shows results of prototyping an infrastructure that enables the implementation of valid self-reconfigurable SDR layouts. It also suggests a design flow useful to implement, in a semi-automated way, SDRs that employ the proposed architecture and infrastructure. On top of the architecture is implemented a software configuration controller, with structure based on a previous model, validated in hardware, the RSCM. The new controller, RSCM-S, was prototyped in Xilinx FPGAs. Comparison data between RSCM and RSCM-S are provided. Several case studies were implemented to demonstrate the viability of the propositions, including a reconfigurable cryptography practical application.

# Sumário

# Lista de Tabelas

# Lista de Figuras

# Lista de Símbolos e Abreviaturas

| | | |
|---|---|---|
| **AC** | Autoconfigurador | 69 |
| **API** | Application Programming Interface | 39 |
| **ASCII** | American Standard Code for Information Interchange | 60 |
| **ASICs** | Application Specific Integrated Circuits | 1 |
| **ASIPs** | Application Specific Instrution Set Processors | 2 |
| **BERT** | Bit Error Rate Tester | 48 |
| **BRAM** | Block RAM | 18 |
| **CCC** | Controle Central de Configurações | 69 |
| **CI** | Circuito Integrado | 1 |
| **CLB** | Configurable Logic Block | 18 |
| **DCR** | Device Control Register | 48 |
| **DDR** | Double Data Rate | 49 |
| **DSP** | Digital Signal Processing | 20 |
| **EC** | Escalonador de Configurações | 67 |
| **EDK** | Embedded Development Kit | 46 |
| **HAL** | Hardware Abstraction Layer | 39 |
| **HDL** | Hardware Description Language | 3 |
| **ICAP** | Internal Configuration Access Port | 14 |
| **IP** | Intellectual Property | 2 |

| | | |
|---|---|---|
| **ISE** | Integrated Software Environment | 46 |
| **LUT** | Look Up Table | 23 |
| **MC** | Memória de Configurações | 69 |
| **MHS** | Microprocessor Hardware Specification | 52 |
| **MR** | Monitor de Reconfiguração | 67 |
| **NCD** | Native Circuit Description | 54 |
| **OPB** | On-chip Peripheral Bus | 48 |
| **PAR** | Place And Route | 23 |
| **PCI** | Peripheral Component Interconnect | 40 |
| **PLB** | Processor Local Bus | 48 |
| **RISC** | Reduced Instruction Set Computer | 47 |
| **RSCM-S** | Reconfigurable System Configuration Manager in Software | 65 |
| **SDR** | Sistema Dinamicamente Reconfigurável | 6 |
| **SDRAMs** | Synchronous Dynamic RAMs | 49 |
| **SoC** | Systems-on-Chip | 2 |
| **SRAMs** | Static RAMs | 49 |
| **SRs** | Sistemas Reconfiguráveis | 3 |
| **TAR** | Tabela de Alocação de Recursos | 76 |
| **TDD** | Tabela de Dependência e Descritores | 67 |
| **UART** | Universal Asynchronous Receiver-Transmitter | 61 |
| **UCP** | Unidade Central de Processamento | 65 |

# Capítulo 1

# Introdução

A indústria de semicondutores moderna é caracterizada por uma rápida evolução que se reflete, por exemplo, no aumento vertiginoso do número de componentes passível de inserção em um único circuito integrado (CI) . Este fato pode ser corroborado pela essência da chamada *Lei de Moore* [41], enunciada na década de 60. Esta prediz, entre outros fatos, a duplicação do número de componentes de um CI no estado da arte da tecnologia a cada 18 meses. Atualmente, não é incomum o emprego de CIs contendo dezenas ou centenas de milhões de transistores em produtos de consumo de baixo custo, tais como *palmtops*, leitores de DVD e jogos eletrônicos. Um problema inerente de usar as tecnologias modernas de implementação de CIs reside na complexidade de projeto de tais componentes.

Tradicionalmente, CIs complexos vendidos em grande volume são projetados de forma personalizada, constituindo circuitos integrados específicos para uma dada aplicação (em inglês, Application Specific Integrated Circuits, ASICs). À medida que a tecnologia evolui, torna-se cada vez menos viável desenvolver CIs de grande densidade com técnicas de projeto tradicionais de ASICs. Alternativas mais eficientes de projeto vêm sendo propostas pela comunidade científica, incluindo abordagens do tipo multiprocessadores homogêneos integrados em um CI, mar de processadores heterogêneos e hardware reconfigurável. A intenção por trás destas abordagens é aumentar a velocidade de projeto, mesmo que com alguma perda de desempenho em relação a ASICs. O raciocínio que justifica estas abordagens é que as pressões de mercado apontam para tempos de projeto menores como a principal forma de dominar o panorama econômico de aplicações tecnológicas.

Na tentativa de adequar o projeto de SoCs às exigências do mercado, uma das abordagens propostas emprega processadores com conjunto de instruções específicos para um aplicação (em

inglês, Application Specific Instrution Set Processors, ASIPs) ao invés de usar hardware específico (ASICs), reduzindo com isto o tempo de projeto e gastos com recursos não recorrentes de engenharia. Atualmente, SoCs projetados com esta abordagem empregam um conjunto heterogêneo de 10 e 15 processadores interligados por redes intrachip configuráveis, tendendo no futuro usar um *mar de processadores* interligados em rede, segundo Henkel em [17].

Outra abordagem empregada no projeto de SoCs é o uso de hardware reconfigurável, ou seja, o uso de plataformas de prototipação rápida onde se pode configurar o hardware desta para uma aplicação específica. Com esta abordagem visa-se reduzir o tempo de projeto obtendo-se produtos com maior vida útil e desempenhos compatíveis ou pouco inferiores a ASICs, em tempo adequado ao lançamento no mercado.

A flexibilidade apresentada por hardware reconfigurável o torna semelhante a sistemas em software, fato que sugere o desenvolvimento de sistemas de suporte a hardware reconfigurável que desempenhem tarefas semelhantes aos sistemas operacionais, tais como escalonadores de tarefas. Este trabalho aborda algumas das tarefas de implementação de infra-estrutura necessárias para habilitar a construção de sistemas de hardware configurável.

O restante deste Capítulo organiza-se da seguinte forma. Na Seção 1.1 são apresentados conceitos básicos sobre sistemas digitais complexos. Na Seção 1.2 são apresentadas definições, classificações e uma discussão sobre vantagens e desvantagens de sistemas reconfiguráveis. A Seção 1.3 apresenta um modelo genérico de sistema reconfigurável conforme proposto por Carvalho em [7]. A Seção 1.4 endereça os motivos que levaram a realização deste trabalho. A Seção 1.5 apresenta os objetivos estratégicos e específicos perseguidos aqui e lista as contribuições alcançadas. Na Seção 1.6 é apresentada a organização do restante do volume.

## 1.1 Reuso de Módulos de Hardware em SoCs

A indústria de microeletrônica está desenvolvendo circuitos integrados cada vez mais complexos. Esta complexidade resulta da maior capacidade de integração de transistores dentro de uma mesma pastilha de silício. Módulos de hardware pré-projetados, pré-verificados e pré-prototipados, também conhecidos por *Núcleos de Propriedade Intelectual* (em inglês, *Intellectual Property Cores ou IP Cores*) aliados à disponibilidade de plataformas de prototipação rápida para validar projetos de alta complexidade tornam SoCs comercialmente viáveis do ponto de vista do projeto, pois reduzem o tempo de projeto de um produto, bem como os gastos não

recorrentes de engenharia.

Segundo Gupta em [15], núcleos IPs podem ser classificado, segundo a forma de interação entre fornecedor e consumidor destes módulos em: "*soft*", "*firm*"ou "*hard*".

- *Soft IPs*: são os núcleos geralmente descritos em linguagem de descrição de hardware (em inglês, *Hardware Description Language - HDL*), oferecendo flexibilidade e independência de tecnologia. Estes núcleos podem ser modificados e empregados com tecnologias de diferentes fabricantes. Os núcleos *soft*, em geral não são disponibilizados com garantias de atendimento de características temporais.

- *Firm IPs*: são núcleos representados em níveis de abstração mais baixos, tipicamente como *netlists* que são geradas para uma dada tecnologia. Estes núcleos podem ser parcialmente parametrizáveis pelo usuário projetista, permitindo que sua arquitetura seja parcialmente adaptável às necessidades do projeto. Entretanto, como a *netlist* é específica para uma dada tecnologia, existem dificuldades para o uso do componente na implementação de CIs junto a diferentes fabricantes.

- *Hard IPs*: são núcleos 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 de características de temporização. Sua desvantagem é a de não permitir qualquer tipo de modificação ou personalização.

O projeto de SoCs complexos reúne um conjunto numeroso (dezenas ou centenas) de IPs, de forma a reduzir o tempo de projeto de um determinado produto. Desta forma, diferentes SoCs projetados para várias aplicações podem empregar um mesmo IP, prática conhecida como *reuso* de núcleos.

## 1.2 Sistemas Reconfiguráveis

Tradicionalmente, sistemas reconfiguráveis (SRs) são definidos como aqueles implementados sobre *hardware reconfigurável*. Os dispositivos comerciais mais difundidos que possuem hardware reconfigurável são os FPGAs (em inglês, *Field Programmable Gate Arrays*). Os FPGAs, bem como praticamente qualquer dispositivo de hardware reconfigurável, possuem uma memória de controle, denominada *memória de configuração*. O preenchimento da memória de configuração controla os recursos do dispositivo, incluindo a interconexão entre segmentos de fios internos

ao dispositivo, a definição de características de pinos do dispositivo (entrada ou saída, padrão de níveis de tensão, etc) e a função dos blocos lógicos (e.g. definição da portas, conteúdos de tabelas-verdade, definição de tipos de elementos de memória, etc).

Uma *configuração* é uma seqüência de valores que, carregados na memória de configuração define para o dispositivo uma *estrutura virtual* de hardware e um comportamento associado a esta estrutura. FPGAs são dispositivos genéricos, com capacidade para implementar qualquer estrutura/comportamento de hardware digital, desde que dentro dos limites de seus recursos físicos. O termo configuração também é usado para definir o processo de preenchimento da memória de configuração. Dada uma seqüência de valores qualquer, preecher a memória de configuração com esta pode produzir efeitos inesperados, inclusive a danificação do dispositivo. Isto leva ao conceito de uma *configuração válida*, aquela que implementa uma estrutura/comportamento coerente com um projeto.

Em sistemas reconfiguráveis o termo *reconfiguração* define o processo de sobrescrever uma configuração válida residente na memória de configuração de um FPGA com outra configuração válida. Assumindo que a memória de configuração de FPGAs é do tipo SRAM, surge a questão do que existe nesta a partir da alimentação do dispositivo. FPGAs são construídos de tal forma que a *configuração inicial* é não válida, pois a estrutura/comportamento definida por esta não é coerente com nenhum projeto. No entanto ela é *segura*, pois não danifica o dispositivo que habilita o processo de configuração/reconfiguração.

Dados estes conceitos, deriva-se como óbvia a necessidade de uso de dispositivos externos para configurar um FPGA, pelo menos após alimentá-lo [1]. Contudo, após a configuração inicial, poder-se-ia imaginar que parte desta estrutura/comportamento inclua o controle do processo de configuração do próprio dispositivo. Um FPGA assim configurado (ou outro SR que assim opere) é denominado um sistema *auto-reconfigurável*.

### 1.2.1 Densidade Funcional

Wirthlin e Hutchings em [48] propuseram o conceito de *densidade funcional* para representar a taxa de utilização dos recursos de um dispositivo integrado. O conceito de *densidade funcional* é útil para comparar diferentes abordagens tecnológicas para implementar uma aplicação. Por exemplo, Dehon [12], mostra que para algumas aplicações, o uso de FPGAs incorre em

[1] É possível externamente gerar comandos para a parte fixa de FPGAs para garantir que este dispare e controle o processo inicial de configuração ao ser alimentado. Este modo de operação não é alvo deste trabalho.

uma utilização muito mais eficiente (medida em termos de densidade funcional) que o uso de microprocessadores, considerando duas implementações equivalentes em termos de desempenho.

Este conceito foi quantificado como o inverso do produto entre a área ocupada pelo circuito e o tempo de execução do mesmo. A Equação 1.1 define densidade funcional (D).

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

A partir desta Equação, pode-se notar que se dois circuitos de funcionalidade equivalente ocupam a mesma área, mas gastam tempos diferentes para executar, então aquele que gasta mais tempo terá uma densidade funcional menor. No caso em que os dois gastam o mesmo tempo para executar uma dada tarefa, mas ocupam áreas diferentes, apresentará maior densidade funcional aquele que ocupar menor área.

### 1.2.2 Classificação de Sistemas Reconfiguráveis e SDRs

Sistemas reconfiguráveis podem ser classificados segundo vários critérios, como já discutido em [43]. Dentre os critérios e classificações sugeridos por diversos autores, tais como [36], [34], [40] destacam-se dois, úteis para raciocinar sobre os sistemas alvo deste trabalho. Sistemas reconfiguráveis podem ser classificados segundo os critérios "*parcela do sistema alterada*"e "*instante de tempo em que ocorre a reconfiguração*". A Tabela 1.1 apresenta os critérios e a denominação das classes associadas aos respectivos critérios.

Tabela 1.1: Critérios e classificações de sistemas reconfiguráveis.

| Critério | Classificação |
|---|---|
| Parcela do Sistema | Totalmente Reconfiguráveis |
| | Parcialmente Reconfiguráveis |
| Instante de Tempo | Estáticos |
| | Dinâmicos |

**Totalmente Reconfiguráveis**: aqueles onde toda reconfiguração é *completa*, ou seja, preenche toda a memória de configuração do dispositivo ou sistema.

**Parcialmente Reconfiguráveis**: são aqueles onde uma reconfiguração do dispositivo ou sistema não precisa preencher toda a memória de configuração.

**Sistemas Reconfiguráveis Estáticos**: são aqueles em que a execução do processo de reconfiguração implica a suspensão da aplicação executando no dispositivo/sistema reconfigurável

em questão.

**Sistemas Reconfiguráveis Dinâmicos**: são aqueles onde a aplicação pode continuar executando durante o processo de reconfiguração do dispositivo/sistema.

O conceito de reconfiguração dinâmica, também denominado de reconfiguração em tempo de execução (do inglês, *Run-Time Reconfiguration ou RTR*) é importante, devido ao principal problema prático enfrentado por dispositivos/sistemas reconfiguráveis, qual seja, o tempo de reconfiguração. Em dispositivos comerciais, este tempo pode atingir a ordem de milissegundos para uma configuração completa.

Muitas aplicações que poderiam se beneficiar de hardware reconfigurável são inviabilizadas devido a esta *latência de chaveamento* de configurações. Embora existam propostas acadêmicas de dispositivos capazes de reduzir esta latência para a ordem de um ou poucos ciclos de relógio, tais como os DPGAs [11], estes dispositivos não se encontram disponíveis comercialmente. Assim, cada implementação pretendendo se beneficiar do uso de hardware reconfigurável deve incluir um cuidado especial com esta latência durante o projeto. Reconfiguração dinâmica tem o potencial para ocultar ou pelo menos reduzir os efeitos desta latência. Contudo, a reconfiguração dinâmica traz consigo uma nova série de problemas a resolver. Um dos principais é o controle de reconfiguração do sistema, alvo deste trabalho.

No presente trabalho, aborda-se tão somente sistemas ao mesmo tempo parcialmente reconfiguráveis e dinamicamente reconfiguráveis, doravante denominados pela sigla *SDR* . Adicionalmente, assume-se que os SDRs alvo deste trabalho são auto-reconfiguráveis.

### 1.2.3 Sistemas Reconfiguráveis: Vantagens e Desvantagens

Em [40], Sanchez aborda dois problemas críticos encontrados em sistemas reconfiguráveis: a interrupção da execução do sistema durante o processo de reconfiguração e o tempo necessário para reconfigurar o hardware até a estabilização da nova configuração do sistema. Este tempo necessário para que uma nova configuração seja lida de um dispositivo de armazenamento e escrita na memória de configuração do FPGA é denominado *latência de chaveamento* como descrito anteriormente.

Sistemas estaticamente reconfiguráveis não possuem seu processamento comprometido porque o processo de reconfiguração é realizado após o término da execução da configuração. Em sistemas dinamicamente reconfiguráveis (SDRs) estes problemas devem ser previstos durante o projeto. Como o processo de reconfiguração é realizado durante a execução do sistema, a latência de

reconfiguração deve ser considerada, pois o desempenho da aplicação está diretamente ligado à sua latência.

Aplicações que empregam SDRs podem obter um melhor desempenho com relação a uma aplicação desenvolvida em software e executada sobre um processador de uso geral, pois parte desta aplicação é processada sobre um hardware reconfigurável dedicado. Com relação à ASICs, o desempenho de SDRs pode aproximar-se ao desempenho de aplicações desenvolvidas em hardware específico, mas seu desempenho é inferior a estas aplicações. Porém em relação a área, os SDRs utilizam menos área em relação a ASICs devido ao conceito de hardware virtual.

A vantagem da flexibilidade proporcionada pelo hardware reconfigurável está diretamente ligada à latência de reconfiguração, de forma que uma aplicação implementada em hardware reconfigurável obtenha um desempenho superior, por exemplo, a mesma aplicação implementada em software e executada sobre um processador de propósito geral. Portanto o compromisso entre flexibilidade e desempenho deve sempre ser considerado no projeto de sistemas dinâmica e parcialmente reconfiguráveis.

A implementação de sistemas computacionais sobre hardware reconfigurável permite que o produto final lançado no mercado possa ser atualizado, com isto prolongando sua vida útil em relação a produtos fabricados com tecnologia de ASICs.

Através da Tabela 1.2 é possível comparar o emprego de SDRs com relação a outras abordagens de implementação de SoCs.

Tabela 1.2: Comparação entre abordagens de implementação de SoCs.

| **Abordagem** | **Área** | **desempenho** | **Flexibilidade** |
|---|---|---|---|
| ASIC | alto consumo | alto | baixa |
| GPP | baixo consumo | baixo | alta |
| SDR | médio consumo | médio | média |

## 1.3 Modelo Genérico de Sistemas Dinamicamente Reconfiguráveis

Esta Seção tem o objetivo de contextualizar o leitor sobre a essência da estrutura que caracteriza um SoC como sendo um sistema reconfigurável. Aqui são apresentados os possíveis estados em que podem se encontrar uma configuração durante a execução do sistema com uma visão abstrata, visando generalizar o conceito de reconfiguração e escondendo detalhes de projeto que não são relevantes neste momento de estudo. A proposta apresentada foi originalmente sugerida

por Carvalho em [7].

Sistemas reconfiguráveis estão diretamente associados ao uso de dispositivos reconfiguráveis tais como FPGAs (a exemplo das propostas de [31], [25], [10]), SoCs contendo internamente um ou mais blocos de hardware reconfiguráveis (como o FPSlic da Atmel [24]) ou mesmo propostas de arquiteturas reconfiguráveis especiais (tais como o Kress Array [16]). No entanto, a essência da reconfigurabilidade está presente em sistemas que não são naturalmente associados ao termo, tais como um computador ou uma rede de computadores. Carvalho em [7] propôs um modelo genérico de sistemas reconfiguráveis, que captura a essência do conceito e engloba todos os exemplos propostos acima [7], [8].

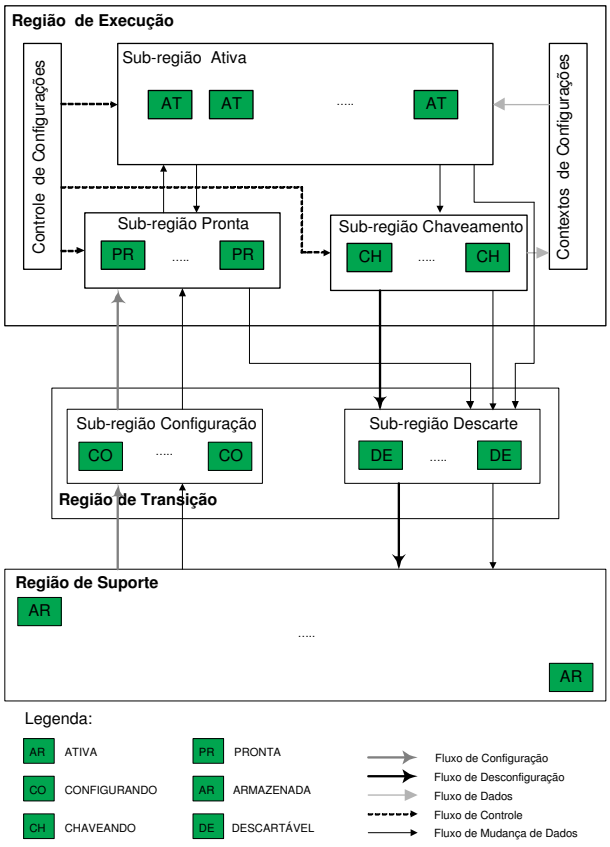


Figura 1.1: Estrutura do modelo GRS - O modelo de Sistema Reconfigurável Genérico é utilizado para representar sistemas reconfiguráveis [7].

A Figura 1.1 ilustra o modelo proposto para representar sistemas reconfiguráveis de forma genérica, modelo este denominado *Sistema Reconfigurável Genérico* (do inglês *Generic Reconfigurable System ou GRS*). Neste modelo, existem três *regiões* que definem o sistema reconfigurável: a região de suporte, a região de transição e a região de execução. Em todas as regiões são en-

contradas *configurações*.

Cada região é definida de acordo com suas funções no sistema reconfigurável. A seguir, é dada uma descrição de cada uma delas.

- *Região de Suporte*: esta região é o repositório de configurações, onde são armazenadas todas as informações que um sistema reconfigurável necessita em algum momento.

- *Região de Transição*: esta região é um repositório onde residem temporariamente configurações em trânsito entre as regiões de execução e de suporte. Esta região encontra-se subdividida em duas sub-regiões. A *Sub-região de Configuração*, na qual encontram-se as configurações que estão sendo inseridas na região de execução e a *Sub-região de Descarte*, que abrange as configurações em processo de desconfiguração ou que estão prontas para serem desconfiguradas.

- *Região de Execução*: é a região correspondente à parte efetivamente operacional do sistema reconfigurável, capaz de realizar trabalho útil. Esta região está subdividida em cinco sub-regiões, sendo duas fixas e três reconfiguráveis. As sub-regiões fixas servem ao controle do processo de reconfiguração do sistema, sendo 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 e que necessitam ser comunicados a outras configurações, após sua desconfiguração. A *Sub-região de Controle de Configuração* 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 a sub-região ativa. A *Sub-região de Chaveamento* agrupa as configurações que se encontram salvando contexto para comunicação com outras configurações.

Neste modelo genérico uma configuração pode encontrar-se em um de seis estados possíveis: Armazenada, Pronta, Ativa, Chaveando, Descartável e Salvando. As possíveis transições de estado estão representadas na Figura 1.2.

No estado *Armazenada* a configuração encontra-se no repositório de configurações. O estado *Configurando* é transitório, sinalizando a ocorrência de um processo de (re)configuração. No

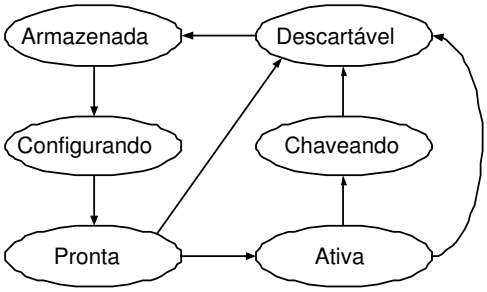


Figura 1.2: Estados de uma configuração segundo o modelo GRS.

estado *Pronta*, a configuração já está implementada no sistema, mas não executando. No estado *Ativa*, a configuração encontra-se em execução. No estado *Chaveando* ocorre o salvamento do contexto da configuração para que possa ser reutilizada. No estado *Descartável* a configuração permanece no sistema, mas não é mais utilizada, podendo ser sobrescrita.

Neste modelo genérico é possível 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"*[7].

Um exemplo de sistema computacional dado por Carvalho [7] e considerações a respeito que o classificam como sistema reconfigurável é apresentado a seguir. Mais exemplos e uma discussão mais extensa destes conceitos encontram-se em [7].

**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, ele 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 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 1.3.

Tabela 1.3: 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 [7].

| **Elemento do modelo** | **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 que 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 |
| Configuração | código executável de um programa |
| Configuração Armazenada | programa em meio de armazenamento secundário |
| Configuração Ativa | programa atualmente de posse do processador, executando |
| Configuração Configurando | programas em processo de carga na memória virtual |
| Configuração Pronta | processo 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 |

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

## 1.4 Motivação

Atualmente, nota-se o aumento do interesse por computação reconfigurável [2], [5], [10], [9], [42], [45], cuja flexibilidade é o fator que pode aumentar a janela de tempo na qual um produto permanece no mercado, ou seja, aumentar a vida útil do produto. A exemplo dos sistemas de software que recebem atualizações para adequar-se aos requisitos do usuário, o hardware implementado em dispositivos reconfiguráveis pode ser adaptado às necessidades de uma aplicação, de forma dinâmica.

Um dos fatores que tornam o uso de SRs uma alternativa interessante é a possibilidade de implementar em uma área reduzida um sistema conceitualmente maior, conceito este denominado *hardware virtual* por Dehon em [13]. A vantagem da virtualização do hardware é a redução do custo do produto final, pois a relação custo/portas lógicas passa a não ser mais linear. Dispositivos com maior número de portas lógicas possuem custo mais elevado que um conjunto de dispositivos de menor capacidade que juntos possuem capacidade similar a dos primeiros.

O desenvolvimento de sistemas reconfiguráveis requer ferramentas que habilitem o uso desta tecnologia e infra-estrutura de suporte para a implementação de SDRs. Um sistema reconfigurável requer elementos que permitam sua implementação incluindo: (i) dispositivos de hardware que permitam reconfiguração parcial em tempo real de execução, (ii) núcleos específicos para controle das operações do sistema [7] e (iii) interfaces padronizadas entre os núcleos que compõem o sistema [33], [6].

Para criar um SDR, necessita-se de ferramentas de projeto para geração automatizada de configurações, de interfaces de comunicação entre partes fixas e reconfiguráveis e ferramentas de verificação para simulação da dinâmica de reconfiguração dos sistemas.

A Xilinx propõem em [60] dois fluxos de projetos para implementação de sistemas dinamicamente reconfiguráveis, sendo eles: o *Fluxo Modular* (em inglês, *Module-Based*) e o *Fluxo por Diferenças* (em inglês *Difference-Based*). No fluxo por diferenças, as configurações parciais são geradas através de cálculos da diferença entre duas configurações muito similares (por exemplo, diferindo apenas em uma dada porta lógica). O fluxo modular, por outro lado, produz configurações parciais que descrevem módulos funcionais complexos do sistema. Este fluxo é considerado complexo e limitado segundo, por exemplo, a análise de Brião em [6].

Para a geração da infra-estrutura básica de um sistema dinamicamente reconfigurável foi proposto o controlador RSCM [7] com a função de gerenciar a execução de configurações em um sistema reconfigurável. Este gerenciador implementado em hardware apresentou limitações,

como por exemplo, não habilitar auto-reconfiguração através de seu emprego, adotar uma política de escalonamento estático, não permitir a preempção e relocação de módulos de hardware e apresentar baixa eficiência de acesso à memória.

## 1.5 Objetivos

O presente trabalho tem como objetivo fundamental propor uma infra-estrutura de hardware/software alternativa à proposta de Carvalho [7] para gerenciar o uso de configurações em SDRs e comparar as duas abordagens. Visa-se aqui suprir algumas das carências encontradas para o desenvolvimento de SDRs, bem como evoluir o modelo do controlador RSCM desenvolvido totalmente em hardware por Carvalho [7]. Pressupõe-se aqui a disponibilidade de um processador com capacidade e disponibilidade para executar processos de controle de configuração. Pressupõe-se também a existência de hardware para executar, em software, a reconfiguração dinâmica do sistema, e que estes recursos são acessíveis através do software do processador.

### 1.5.1 Objetivos Estratégicos

São objetivos estratégicos desta dissertação:

1. Dominar a tecnologia de FPGAs capazes de habilitar aplicações auto-reconfiguráveis.
2. Desenvolver os recursos necessários para estabelecer uma interface de comunicação entre áreas fixa e reconfiguráveis, em SDRs.
3. Dominar as ferramentas de CAD existentes e estabelecer um novo fluxo de projeto para SDRs.

### 1.5.2 Objetivos Específicos

1. Dominar o uso de drivers para desenvolver uma API que permita o controle da auto-reconfiguração em software.
2. Desenvolver uma infra-estrutura de hardware que estabeleça uma interface de comunicação entre áreas fixas e reconfiguráveis.
3. Definir e implementar *em software* as funções do controlador de configurações com base na proposta do controlador RSCM de Carvalho [7].

4. Validar o controlador de configurações implementado.

5. Medir os tempos de reconfiguração, área de ambos os controladores de configuração.

6. Comparar o desempenho de ambos controladores de acordo com os resultados das medições realizadas.

### 1.5.3 Contribuições

Com a realização deste trabalho, foram obtidas as seguintes contribuições:

- Domínio do funcionamento da porta interna de acesso a configurações (do inglês, *Internal Configuration Access Port* ou ICAP), que habilitam a auto-reconfiguração em dispositivos Xilinx.

- Disponibilização de macros para comunicação entre IPs fixos e reconfiguráveis.

- Domínio do processo de projeto da comunicação entre processador e IPs reconfiguráveis através de interrupções.

- Domínio do processo de projeto de infra-estruturas de hardware com uma ou duas áreas reconfiguráveis com suporte a auto-reconfiguração.

- Disponibilização de um módulo desenvolvido em software para controle de configurações através da porta ICAP.

- Recursos de software para compactação e descompactação de arquivos de configuração.

- Disponibilização de dois algoritmos para criptografia de dados implementados em hardware, para o desenvolvimento futuro de uma aplicação demonstrando o uso da infra-estrutura para sistemas auto-reconfiguráveis.

## 1.6 Organização do Restante do Volume

O restante deste documento encontra-se distribuído em 5 Capítulos descritos a seguir.

No Capítulo 2, são apresentadas definições de conceitos que servem como base para o entendimento e contextualização deste trabalho.

No Capítulo 3, apresentam-se informações relacionadas ao estado da arte em pesquisa sobre infra-estrutura de hardware para SDRs, bem como propostas e implementações de estruturas para o controle de configuração em SDRs.

No Capítulo 4, apresenta-se uma descrição das ferramentas de CAD utilizadas para o desenvolvimento da infra-estrutura de hardware e a definição de um novo fluxo de projeto para SDRs, alternativa ao fluxo de projetos proposto pela empresa Xilinx.

O Capítulo 5 apresenta a proposta e dados de implementação do controlador de configuração em software.

O Capítulo 6 apresenta os estudos de caso desenvolvidos com o intuito de validar a implementação da infra-estrutura de hardware e do controlador de configurações. Este Capítulo apresenta e discute as medições realizadas para a comparação das implementações.

No Capítulo 7 são apresentadas as conclusões obtidas com a realização deste trabalho, listando um resumo de contribuições e destacando possíveis propostas de trabalhos futuros.

# Capítulo 2

# Aspectos Relacionados à Infra-estrutura de SDRs

O desenvolvimento de sistemas reconfiguráveis conduz ao projeto de hardware com flexibilidade semelhante ao projeto de software. Desta forma, o hardware reconfigurável necessita de subsistemas que executem operações semelhantes às de um sistema operacional. Um sistema operacional típico gerencia processos em um processador, controlando a carga de programas a executar em memória. Para isto, deve seguir uma política proveniente de um escalonador de processos. Em SDRs, deve existir um subsistema que realize tarefas semelhantes a esta.

Para a implementação de qualquer sistema computacional complexo, é necessário suporte de hardware e software. No caso de SDRs, o suporte de hardware pode tomar a forma de plataformas de prototipação entre outros meios, enquanto o software é representado por ferramentas de projeto. Como já descrito anteriormente, existe carência de ferramentas de suporte ao desenvolvimento de SDRs. Além disso, as existentes não satisfazem as reais necessidades dos projetistas, como a automatização do projeto de sistemas dessa natureza para que os mesmos possam respeitar as restrições impostas pelo mercado com relação ao tempo para o produto chegar ao mercado (em inglês, *time to market*).

O projeto de SDRs baseados em FPGAs deve considerar as restrições da arquitetura destes dispositivos e das ferramentas disponíveis para o desenvolvimento de sistemas reconfiguráveis. Nesta Seção serão apresentados aspectos relacionados ao projeto da infra-estrutura de hardware de SDRs. A seguir são listados os pontos relevantes no projeto:

- Tecnologias de suporte a reconfiguração parcial;

- Definição do layout da arquitetura;
- Fluxos de projetos;
- Geração de arquivos de configuração;
- Armazenamento e compactação de configurações parciais;

## 2.1 Arquiteturas de FPGAs com Suporte à Reconfiguração Parcial

Poucas famílias de dispositivos comerciais suportam reconfiguração dinâmica e parcial. Segundo [27], os primeiros dispositivos que suportaram reconfiguração parcial são: o *Clay*, fabricado pela National Semicondutors [30], o *Cal1024*, fabricado pela *Algotronix* [1] e o *XC6200*, fabricado pela Xilinx [49]. A família XC6200 foi usada como dispositivo alvo para a modelagem de várias propostas de controladores de configurações. Aparentemente, devido ao suporte precário de ferramentas de software para projeto, estes FPGAs não tiveram sucesso comercial, embora tenham sido utilizados amplamente no meio acadêmico.

Atualmente, dois fabricantes comercializam dispositivos que habilitam reconfiguração dinâmica e parcial. São eles, a *Atmel* e a *Xilinx*. A Atmel fabrica duas famílias que possibilitam reconfiguração dinâmica e parcial. São elas a AT6000 e AT40k. A série AT40k [3] 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 [4].

A Xilinx comercializa hoje várias famílias que suportam reconfiguração dinâmica e parcial, sendo elas Spartan2/2G, Spartan3/3E, Virtex, Virtex E, VirtexII, VirtexII-Pro e VirtexIV. Os FPGAs das famílias 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. A Figura 2.1 apresenta a organização interna de um dispositivo VirtexII-Pro utilizado neste trabalho, diferente da família VirtexII apenas por conter núcleos processadores PowerPC embarcados, na forma de *hard cores*.

A memória de configuração em dispositivos Xilinx é organizada como uma matriz bidimensional de bits. Estes bits são agrupados em *quadros verticais* (em inglês, *frames*) com um bit de largura, e se estendem verticalmente na forma de colunas que atravessam todo o dispositivo.

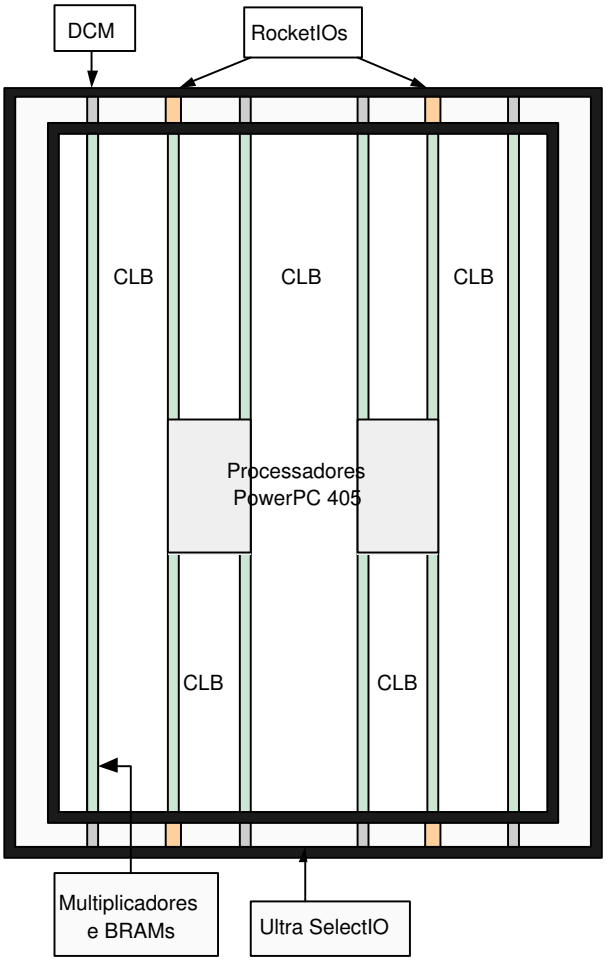


Figura 2.1: Organização interna de um dispositivo VirtexII - Pro da Xilinx.

Um quadro é 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 (CLBs), memórias, I/Os, multiplicadores) é dividida em um certo número de quadros.

Os FPGAs das famílias VirtexII, VirtexII-Pro e VirtexIV [50], além dos componentes básicos existente nas famílias Virtex e Spartan2/2G, possuem colunas de multiplicadores de $18 \times 18$ bits e a porta interna de acesso a configuração (ICAP). Mais informações sobre estas famílias podem ser encontradas em [51]. Os dispositivos da família VirtexII-Pro [52] constituem uma expansão da família VirtexII. Cada dispositivo desta família (com excessão ao dispositivo XC2VP2 que não possui processadores embarcados) contém pelo menos 1 processador IBM PowerPC405 embarcado.

Os FPGAs da família VirtexIV estão divididos em três classes baseadas na aplicação alvo: LX, FX e SX. As plataformas LX são indicadas para aplicações com maior demanda de área de

prototipação. As plataformas FX são endereçadas a aplicações que exigem maior quantidade de recursos para desenvolvimento de sistemas embarcados. As plataformas SX são indicadas para aplicações em processamento digital de sinais (em inglês, Digital Signal Processing ou DSP) . Pode-se imaginar que os FPGAs VirtexIV LX/SX são evoluções da família VirtexII, enquanto que os FPGAs VirtexIV FX são uma evolução da família VirtexII-Pro.

A Tabela 2.1 apresenta um quadro comparativo de alguns dos dispositivos que habilitam reconfiguração dinâmica e parcial. Nesse comparativo, pode-se perceber que os dispositivos das famílias Virtex, VirtexII, VirtexII-Pro e VirtexIV comercializados pela Xilinx apresentam uma densidade significativamente maior que os dispositivos Atmel. As densidades dos dispositivos da família VirtexII-Pro/VirtexIV foram obtidos através de estimativas sem considerar os processadores PowerPC embarcados.

A plataforma alvo para implementação dos sistemas propostos neste trabalho é composta pelo dispositivo da família VirtexII-Pro fabricado pela Xilinx, devido ao fato desta plataforma:

- encontrar-se disponível no ambiente onde o trabalho foi desenvolvido;
- possuir o dispositivo ICAP.

A plataforma de prototipação VirtexII-Pro$^{TM}$ FF1152 da Memec-Insight [26] usada nesse trabalho contém outros recursos que propiciam ao usuário a oportunidade de desenvolver sistemas de diferentes naturezas, conforme ilustrado na Figura 2.2. Os recursos encontrados são:

- 3 fontes de relógios (25 a 700MHz);
- oscilador LVTTL de 100MHz;
- 2 blocos de memória SDRAM de 32MB;
- expansão para interface P160;
- 40 pinos de entrada e saída para manipulação do usuário/projetista;
- porta JTAG para configuração
- 2 portas seriais;
- 8 LEDs para manipulação do usuário/projetista;
- 8 chaves DIP e 3 *Push-Buttons*;

Tabela 2.1: Comparação entre as propostas de controladores.

| Família | Dispositivo | Quantidade de Portas Lógicas Equivalentes |
|---|---|---|
| AT40k | AT40K05 | 5k - 10k |
| | ... | ... |
| | AT40K40 | 40k - 50k |
| AT6000 | AT6002 | 6k |
| | ... | ... |
| | AT6010 | 30k |
| Virtex | XCV50 | 50k |
| | ... | ... |
| | XCV1000 | 1M |
| VirtexII | XC2V40 | 40k |
| | ... | ... |
| | XC2V8000 | 8M |
| VirtexII-Pro | XC2VP2 | 80k |
| | XC2VP30 | 3M |
| | ... | ... |
| | XC2VP100 | 10M |
| VirtexIV | XC4VLX15 | 1,5M |
| | XC4VLX200 | 20M |
| | XC4VSX25 | 2.5M |
| | XC4VSX55 | 5,5M |
| | XC4VFX12 | 1,2M |
| | XC4VFX140 | 14M |

## 2.2 Definição da Estrutura Física de um SDR

Tipicamente, para implementar SDRs é necessário organizar a arquitetura do hardware do sistema de maneira que a parte fixa e as partes reconfiguráveis sejam capazes de se comunicar. A implementação da interface física entre a parte fixa do SDR e as partes reconfiguráveis deve ser tal que duas ou mais partes reconfiguráveis distintas que possam usar a mesma região do FPGA em momentos distintos devem: (i) *usar o mesmo protocolo de comunicação* e (ii) a *mesma interface física*. Isto visa tornar a comunicação da parte fixa com as partes reconfiguráveis independente da parte reconfigurável presente a cada instante.

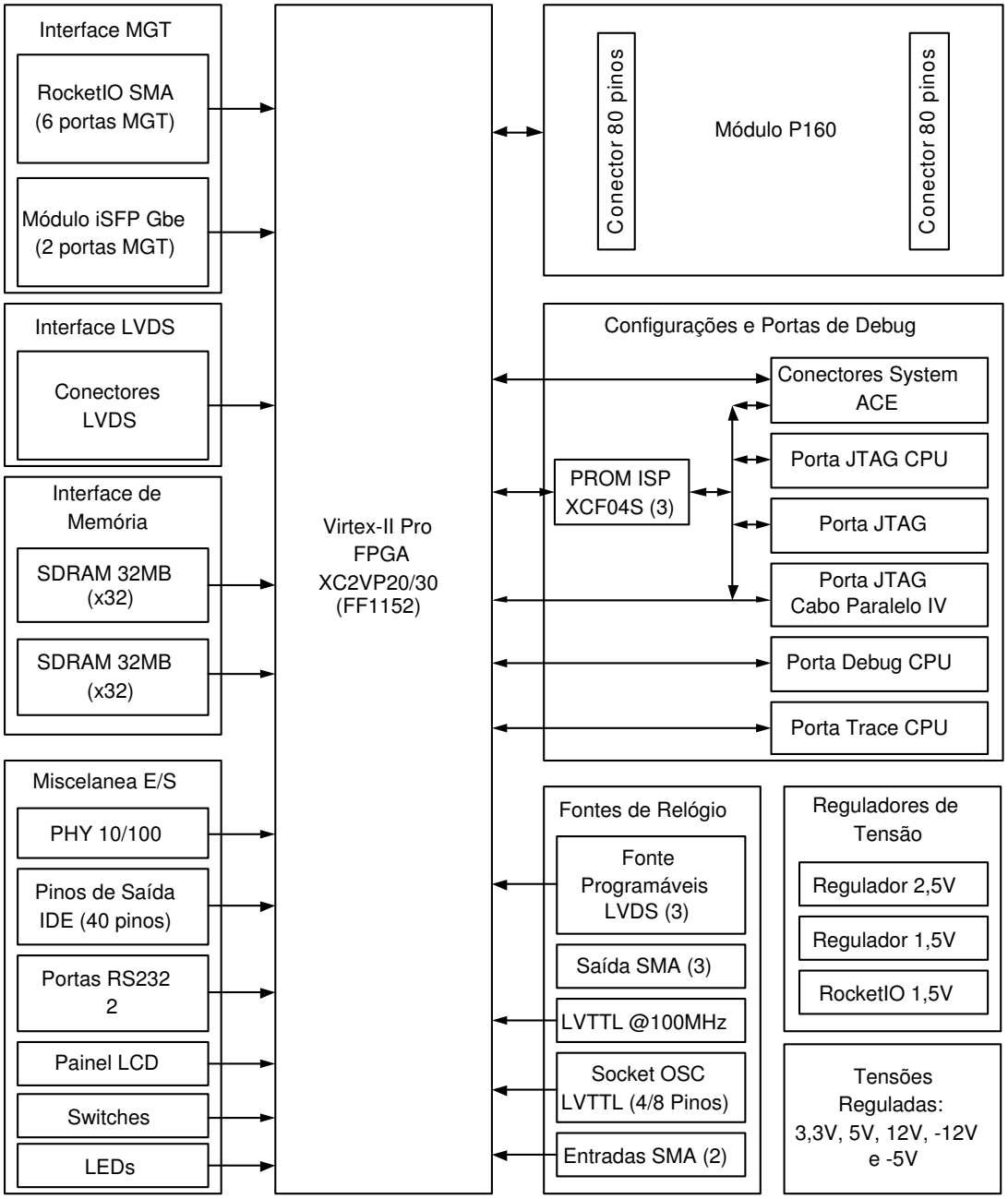


Figura 2.2: Plataforma de prototipação disponível: VirtexII-Pro$^{TM}$ FF1152 da Memec-Insight [26].

Para garantir que a condição (i) seja atendida, basta realizar os projetos da parte fixa e das partes reconfiguráveis usando uma mesma interface (i.e., mesmos sinais e mesmo protocolo de comunicação). A condição (ii), contudo é mais complexa, pois implica controlar a definição da comunicação em termos de posição que ocupa cada fio no leiaute físico final do FPGA. Em FPGAs Xilinx, não é possível restringir a posição de fios durante o roteamento, uma limitação do

software *PAR* (do inglês, *Place and Route*)[53]. Assim, deve-se recorrer a artifícios para garantir a posição física da interface no leiaute.

O artifício usado é definir a posição física da interface mediante o uso de blocos lógicos pré-posicionados e pré-configurados para funcionar como ponto de passagem de fios. Isto pode ser feito usando LUTs do FPGA configuradas como funções identidade, como proposto por Möller [29] ou utilizando *tri-states* do próprio FPGA conforme proposto pela Xilinx, posicionados no local onde se deseja um pino de interface. Estas estruturas pré-definidas são denominadas *Macros*. A Figura 2.3 apresenta uma ilustração da restrição encontrada no software para definir o posicionamento de fios da interface de comunicação entre partes fixas e reconfiguráveis. Com esta Figura percebe-se a necessidade do uso de macros para satisfazer a condição (II) da infra-estrutura de SDRs.

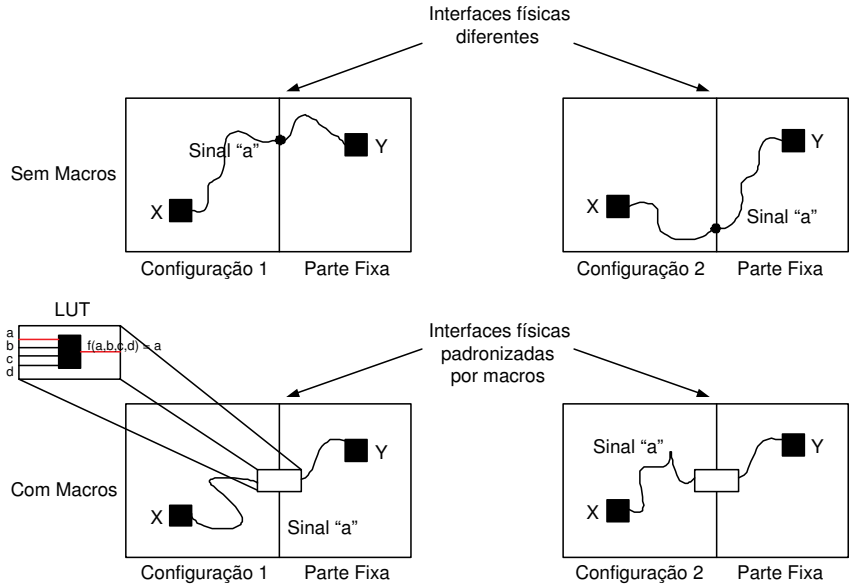


Figura 2.3: Problema encontrado para fixar pinos de comunicação entre partes fixas e reconfiguráveis. Uma proposta de solução é o uso de macros compostas por LUTs com função identidade.

Durante o processo de reconfiguração, o hardware presente na área reconfigurada pode apresentar um comportamento imprevisível, gerando sinais espúrios para a parte fixa causando uma instabilidade do sistema. Para prevenir isto, as macros também realizam o isolamento das partes fixa e reconfiguráveis evitando a desestabilização do sistema durante o processo de reconfiguração parcial.

Para satisfazer as condições de comunicação e isolamento apontadas, é necessário estabelecer uma estratégia, levando em conta a tecnologia empregada, para posicionar fisicamente sobre a

matriz de CLBs do FPGA os módulos que compõem a arquitetura do sistema. Para garantir um bom leiaute do sistema, o projeto deve atender a seguinte estratégia:

1. um módulo reconfigurável não deve compartilhar uma coluna de CLB com outro módulo da arquitetura;
2. o posicionamento das macros deve garantir que ambas as partes fixas e reconfiguráveis sejam isoladas;
3. as macros devem estar posicionadas sobre as mesmas colunas de CLBs para que a interface seja "linear";
4. preferencialmente as macros devem garantir que fios pertencentes a uma dada área ultrapassem seus limites apenas através das macros.

A etapa de definição de leiaute para comunicação e isolamento de áreas é definida como *planta baixa* da arquitetura do sistema (em inglês, *floorplanning*). Durante a realização da planta baixa, os módulos que compõem a arquitetura são considerados como figuras geométricas retangulares, cuja área é diretamente proporcional à complexidade do módulo.

A Figura 2.4 representa o exemplo de uma arquitetura genérica para SDRs com $n$ áreas reconfiguráveis, uma área fixa e a interface de comunicação presente entre elas.

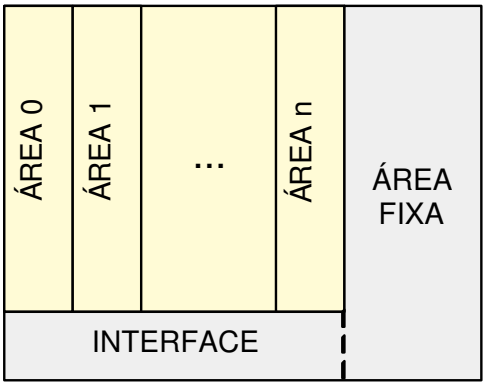


Figura 2.4: Exemplo de planta baixa para o projeto de sistemas parcialmente reconfiguráveis.

Os módulos referenciados como $Área_0$ à $Área_n$ indicados na Figura 2.4 são as partes reconfiguráveis do sistema e suas configurações são alteradas de acordo com a execução do sistema. A parte fixa representa o controle do sistema e detém a política de substituição de configurações das áreas reconfiguráveis. A *Interface* representa o meio que suporta a comunicação entre as partes fixa e reconfiguráveis.

## 2.3 Fluxos de Projeto

Um dos fabricantes de dispositivos FPGAs e de ferramentas de projeto para sistemas embarcados, Xilinx, sugere dois fluxos de projetos para realização de reconfiguração parcial em [60]. O primeiro fluxo endereça reconfigurações onde apenas alguns bits são alterados na lógica do sistema, fluxo conhecido como *Diferencial*. O segundo fluxo endereça reconfigurações de partes do sistema, sendo conhecido como fluxo *Modular*.

A Figura 2.5 apresenta as etapas do fluxo modular proposta pela Xilinx.

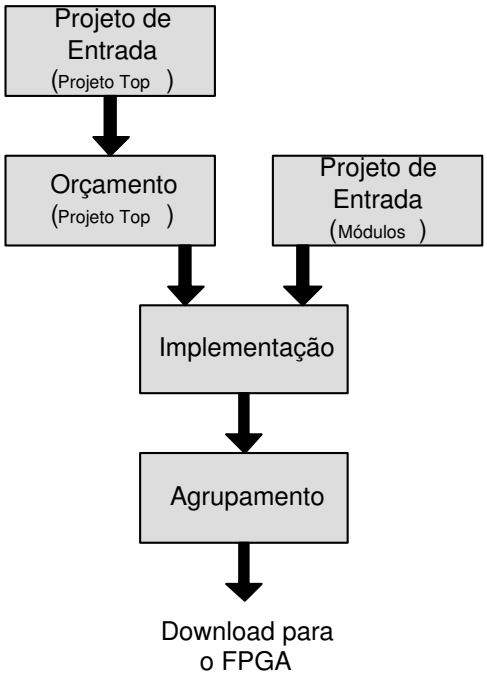


Figura 2.5: Fluxo de projeto modular proposto pela Xilinx.

**Entrada do Projeto e Síntese:** projetistas elaboram módulos através de HDLs. Nesta etapa, são criados o projeto *top* que contém a descrição completa da arquitetura e os módulos funcionais que compõem o sistema e podem ser substituídos em tempo de execução.

**Orçamento:** nesta fase o projetista realiza o posicionamento dos módulos que compõem o sistema e prevê o consumo de recursos do FPGA para implementar o sistema.

**Implementação:** nesta etapa, o projetista implementa os módulos do sistema, gerando as configurações para o FPGA.

**Agrupamento:** nesta fase, os projetistas reúnem os módulos segundo a estrutura definida no projeto Top.

No fluxo modular existe a preocupação com a comunicação entre os módulos móveis e os módulos fixos do sistema. Como diferentes configurações podem ocupar a mesma área reconfigurável, estas devem apresentar interfaces de comunicação com o mesmo padrão, ou seja, com os mesmos sinais e mesmo posicionamento, evitando problemas de incompatibilidade durante a execução do sistema. Outro fator importante também relacionado com a comunicação entre módulos de um sistema é a possibilidade de geração de sinais espúrios durante o processo de reconfiguração. Estes sinais podem causar falhas na execução do sistema de controle, tornando necessário o isolamento elétrico da área no instante em que será reconfigurada.

Em virtude disto, a Xilinx propôs o uso de uma estrutura chamada *Macro* capaz de estabelecer a comunicação entre módulos permitindo ainda o isolamento elétrico através de sinais de controle. A Macro é um módulo composto por quatro fios para comunicação e outros quatro para controle de fluxo. Sistemas onde a interface de comunicação precise de mais fios podem utilizar várias Macros conectadas paralelamente. A Figura 2.6 apresenta a estrutura de uma Macro proposta pela Xilinx.

Com o uso de Macros, é possível formar uma interface de comunicação entre áreas fixa e reconfiguráveis com um isolamento elétrico entre elas.

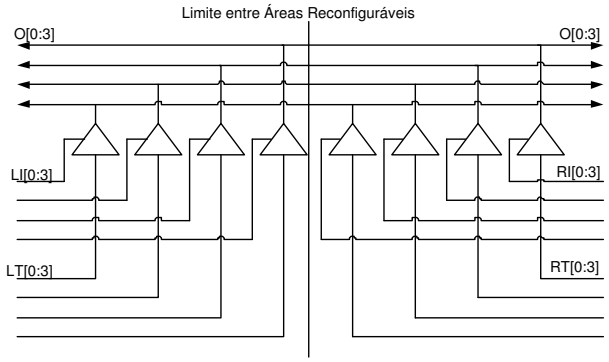


Figura 2.6: Macro desenvolvida pela Xilinx permitindo a comunicação entre áreas fixa e reconfiguráveis e o isolamento elétrico entre elas a para compor a interface de comunicação entre áreas fixas e reconfiguráveis.

## 2.4 Geração de Arquivos de Configuração

O projeto de sistemas em hardware reconfigurável está diretamente associado ao uso de FPGAS e na tecnologia empregada nestes dispositivos. Os fabricantes de FPGAs oferecem

ferramentas em software para desenvolver sistemas embarcados. Estas ferramentas obtêm como entrada uma descrição da arquitetura de hardware do sistema e geram como saída arquivos com as informações necessárias ao preenchimento da memória de configuração de um FPGA específico.

As ferramentas em software que realizam este tipo de processamentos são denominadas *ferramentas de síntese* e os arquivos de saída obtidos como resultado da síntese são denominados *bitstreams*. Estes arquivos contêm informações sobre a configuração de todo o sistema, caso em que são denominados *bitstreams totais* ou apenas parte do sistema, caso em que são denominados *bitstreams parciais*. O funcionamento das ferramentas de síntese é detalhado no Capítulo 4, onde são apresentados os ambientes de desenvolvimento utilizados neste trabalho.

## 2.5 Armazenamento e Compactação de Arquivos de Configuração

De acordo com o modelo genérico de SDRs proposto por Carvalho em [7] e apresentado no Capítulo 1, uma configuração pode encontrar-se armazenada (estado inativa) em um repositório de dados enquanto o sistema não a requisitar. Portanto, este repositório deve ter capacidade de armazenamento suficiente para todas as configurações inativas do sistema. Em sistemas auto-reconfiguráveis o repositório deve ser a memória disponível na plataforma de prototipação, para que o próprio sistema tenha condições de gerenciar as configurações.

A capacidade de memória disponível em plataformas de prototipação pode ser insuficiente para armazenar todos os arquivos de configurações inativas do sistema. Neste caso torna-se necessário o uso de compactação dos bitstreams. Com isto, os arquivos devem ser compactados durante o projeto e armazenados em memória no momento da prototipação do sistema. O sistema deve utilizar um módulo descompactador de bitstreams executado durante o processo de reconfiguração. O uso deste módulo aumenta a latência de reconfiguração, porém otimiza a ocupação da memória permitindo armazenar um número maior de configurações. Este compromisso entre latência e ocupação da memória deve ser previsto no projeto do sistema.

# Capítulo 3

# Estado da Arte

O projeto de SDRs, em particular, de sistemas auto-reconfiguráveis, deve prever a construção de uma infra-estrutura de hardware que suporte a implementação destes sistemas, bem como a construção de um subsistema em hardware ou software que gerencie a dinâmica de substituição do hardware reconfigurável.

SDRs necessitam de uma infra-estrutura de hardware que atenda exigências tais como: a definição de uma interface de comunicação entre hardware fixo e hardware reconfigurável (móvel), isolamento entre as partes reconfiguráveis e o resto do circuito de forma que o hardware fixo não tenha seu processamento interrompido durante o processo de reconfiguração e tecnologia que suporte a implementação destes sistemas.

A flexibilidade dos SDRs deve ser gerenciada por um subsistema semelhante ao sistema operacional, conforme dito anteriormente. Este subsistema deve apresentar algumas funções básicas. Um exemplo de tarefa é identificar pontos no processamento onde seja necessária a reconfiguração de uma nova tarefa em hardware. Este subsistema deve ainda possuir um escalonador de configurações que estabeleça uma política de substituição das configurações disponíveis no sistema.

Neste Capítulo serão revisadas algumas propostas de infra-estrutura de hardware para SDRs na Seção 3.1. A Seção 3.2 discute propostas de controladores de configurações. A Seção 3.3 apresenta uma discussão sobre as propostas revisadas, destacando vantagens e desvantagens dos modelos e implementações.

## 3.1 Propostas de Infraestrutura de Hardware para SDRs

O desenvolvimento de sistemas que utilizam hardware parcialmente reconfigurável requer cuidado com pontos críticos de projetos, como por exemplo: (i) interface de comunicação entre módulos reconfiguráveis e módulos fixos, (ii) número de pinos que constituem a interface de comunicação e (iii) tamanho da área reconfigurável compatível com o tamanho do módulo reconfigurável.

### 3.1.1 Proposta de Palma et al.

Palma et al. em [35] propõem uma solução ao problema da interface de comunicação entre módulos reconfiguráveis e fixo. Segundo Palma et al. [35] em sistemas dinâmicos e parcialmente reconfiguráveis deve existir uma interface de comunicação entre IPs que permita inserção e remoção de núcleos IPs sem a interrupção do processamento do sistema. Com este propósito os autores propõem uma interface de comunicação entre módulos reconfiguráveis e fixo.

Esta proposta visa construir uma interface fixa de comunicação que realize três funções fundamentais: árbitro do barramento, comunicação entre módulos e virtualização de pinos de entrada e saída (E/S). A Figura 3.1 apresenta a estrutura da interface de comunicação proposta em [35].

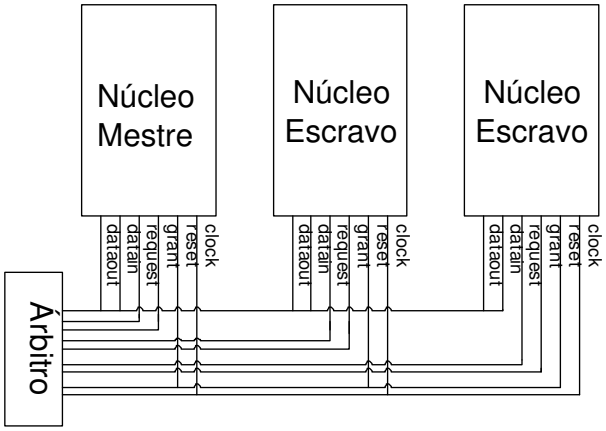


Figura 3.1: Interface de comunicação proposta por Palma et al. [35].

- árbitro do barramento: tem o objetivo de garantir a restrição de acesso ao barramento evitando conflitos de acesso.
- Comunicação: a principal função da interface é estabelecer um meio de comunicação entre os núcleos IPs.

- Virtualização de pinos (E/S): a virtualização de pinos significa que a interconexão dos pinos de entrada e saída dos IPs e pinos exteriores do FPGA são realizadas através da interface de comunicação, e da parte fixa do SDR.

Para implementar o barramento proposto utilizam-se buffers tri-states nas entradas e saídas das áreas reconfiguráveis. Isso se deve ao fato das linhas de tri-states apresentarem um roteamento fixo entre CLBs vizinhas em dispositivos Virtex. As duas camadas de tri-states são necessárias porque uma delas pertence ao módulo reconfigurável e outra pertence ao barramento. Na abordagem de Palma et al., o barramento localiza-se na parte inferior do dispositivo, como componente isolador entre as áreas reconfiguráveis, como visto na Figura 3.1. Devido à restrição do número de buffers tri-state nos FPGAs Xilinx utiliza-se comunicação serial entre os IPs com o intuito de reduzir o número de sinais na comunicação na interface.

### 3.1.2 Proposta de Huebner et al.

A proposta de solução de Huebner et al. em [18] baseia-se na construção de uma interface fixa que permita a comunicação entre módulos fixos e reconfiguráveis. Esta interface foi denominada de *Bus Macro* [18]. A Bus Macro proposta é capaz de transportar sinais unidirecionalmente, ou seja, transmitem sinais em apenas um sentido. Para que a comunicação entre os módulos da arquitetura ocorra em ambos sentidos foi proposta a construção de *macros de entrada* e *macros de saída* com o objetivo de formar um barramento de comunicação bidirecional. Huebner et al. propõem uma arquitetura organizada com um projeto específico, de forma que permita aos módulos fixos e reconfiguráveis comunicarem-se através das macros. A Figura 3.2 mostra a estrutura da arquitetura proposta.

A arquitetura implementada sobre FPGAs VirtexII, possui um módulo controlador baseado no processador *MicroBlaze* [58], e dispõe de memória flash externa ao FPGA para armazenar bitstreams parciais que poderão ser usados durante a execução do sistema. A ICAP possibilita a reconfiguração do FPGA sem precisar de elementos externos para este processo. O *Árbitro* gerencia o endereçamento dos módulos, fazendo a transmissão e recepção de dados. Cada módulo reconfigurável possui uma interface chamada *BusCom* responsável por estabelecer a comunicação entre módulo e barramento. Cada *Módulo Bus Com* tem a função de armazenar dados de entrada e sinalizar ao árbitro sobre a disponibilidade de dados de saída. Este módulo contém uma identificação, a qual é usada pelo árbitro para que sejam buscados e enviados dados através das macros. Na Figura 3.2, apresenta-se a estrutura do módulo *Bus Macro*, que corresponde ao

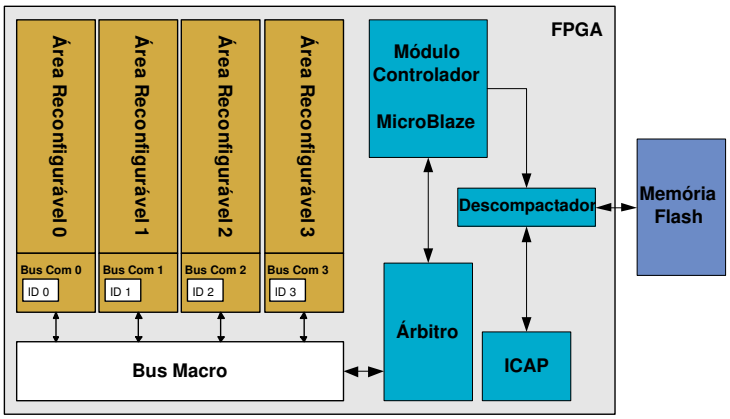


Figura 3.2: Modelo da arquitetura de hardware proposto por Huebner et al. [18].

barramento formado por macros de entrada e saída.

- Macro de Entrada

A arquitetura projetada por Huebner et al. contém 4 módulos reconfiguráveis e um árbitro que estabelece prioridades de comunicação entre o módulo de controle e módulos reconfiguráveis. O diagrama esquemático da macro de entrada definida para esta arquitetura é mostrado na Figura 3.3.

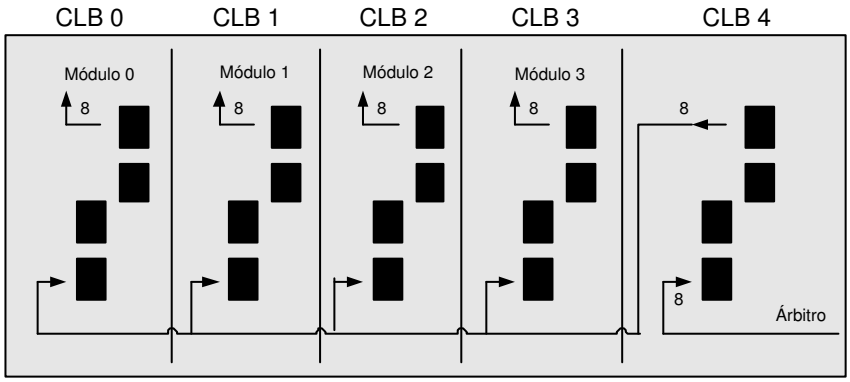


Figura 3.3: Esquemático da macro de entrada de Huebner et al.. Cada CLB é composto por 4 slices (retângulos escuros), cada um contendo 2 LUTs e 2 flip-flops.

A macro de entrada é formada por 5 CLBs, onde 4 CLBs conectam-se aos módulos reconfiguráveis e um CLB ao árbitro. Cada CLB é formado por 4 *Slices*, e cada Slice contém duas tabelas-verdades configuráveis (em inglês, Look Up Tables - LUTs) [50]. As LUTs contém 4 entradas e uma saída de dados. É possível programar as LUTs para transpor dados de entrada

para a saída. Esta estrutura interna da CLB permite que a macro projetada transmita apenas 8 sinais, sendo possível a replicação de macros em paralelo, caso seja necessário aumentar a largura do barramento de dados.

A Figura 3.4 apresenta a implementação da macro de entrada utilizando CLBs do FPGA.

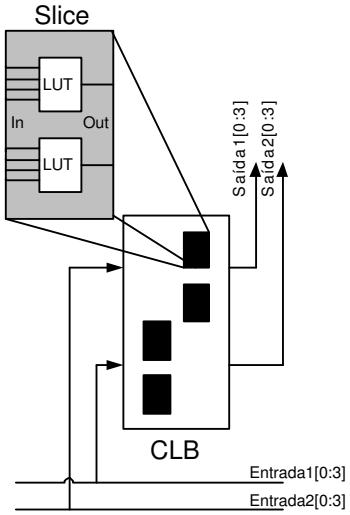


Figura 3.4: CLB configurada como função identidade para atuar como macro de entrada.

- Macro de Saída

Similarmente à macro de entrada, a macro de saída também utiliza 5 CLBs. Entretanto, as LUTs são inicializadas para realizar multiplexação (2:1), como mostrado na Figura 3.5. A multiplexação é necessária para garantir que o acesso para escrita no barramento do sistema seja restrito a apenas um módulo. Quando um módulo escreve dados no barramento, o sinal *select* seleciona a respectiva entrada do multiplexador, de modo que os módulos antecessores não sejam habilitados a usar o barramento. A prioridade de acesso é fixa e faz com que os módulos mais à esquerda não sejam atendidos caso um módulo próximo ao árbitro estiver constantemente solicitando operações de escrita na parte fixa. Esta política garante a restrição de acesso ao barramento. A Figura 3.5 apresenta uma descrição da macro de saída para cada CLB.

Em [19], Huebner et al. propõem desenvolver uma rede de comunicação baseada no uso das macros de[18], visando disponibilizar uma infra-estrutura para o desenvolvimento de sistemas de hardware parcialmente reconfiguráveis.

Os autores propõem construir uma rede de comunicação capaz de adaptar-se à demanda de uma aplicação em tempo de execução. Esta rede utiliza uma política de acesso baseada em fatias

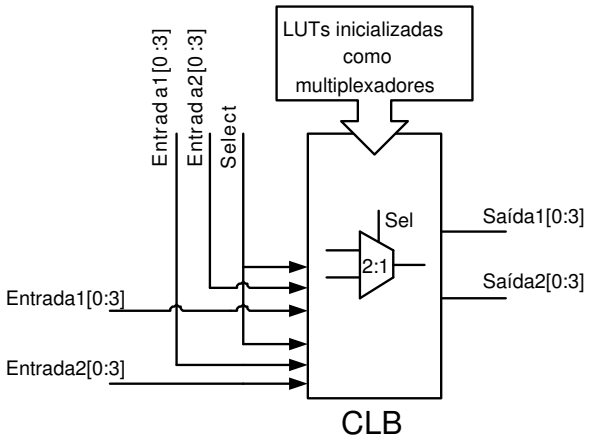


Figura 3.5: Implementação de multiplexadores nas LUTs dos CLBs formando a macro de saída.

de tempo. A Figura 3.6 apresenta um esquemático da estrutura interna da rede de comunicação.

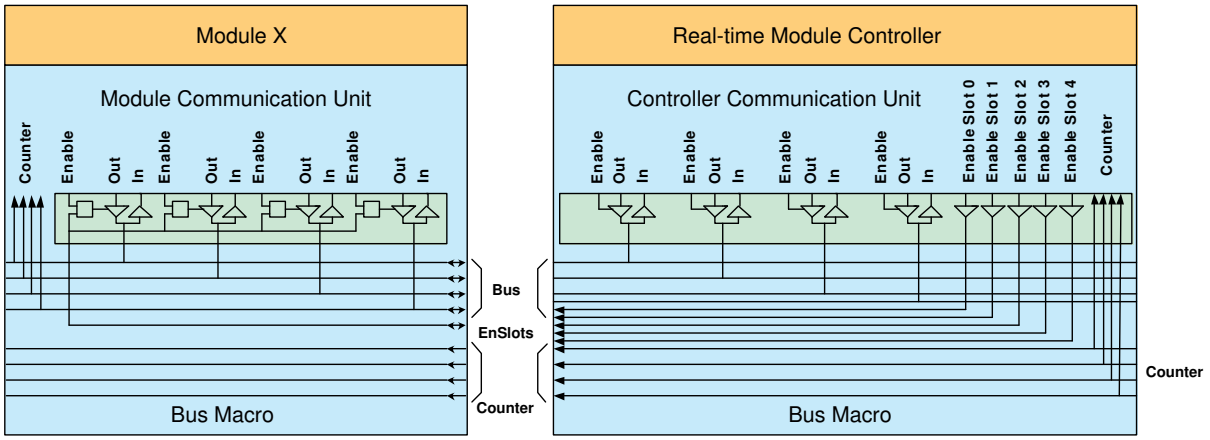


Figura 3.6: Estrutura interna do barramento de comunicação proposto por Huebner et al. [19].

Esta rede de comunicação é projetada para operar na arquitetura apresentada na Figura 3.2 em [18]. A rede é constituída por 4 barramentos bidirecionais formados por macros. A cada barramento estão associados um contador de tempo e um sinal de habilitação do barramento. Isto permite que cada um dos módulos acesse a todos barramento em instantes de tempo diferentes.

Os módulos reconfiguráveis e de controle da arquitetura contêm tabelas com valores que indicam as fatias de tempo em que terão direito de acesso ao barramento. Estas tabelas são reconfiguradas durante a execução do sistema de modo a otimizar o compartilhamento da rede de acordo com a demanda da aplicação. A alteração desta tabela faz com que a topologia da rede seja alterada dinamicamente. A Figura 3.7 ilustra um exemplo de uma possível topologia de rede de comunicação.

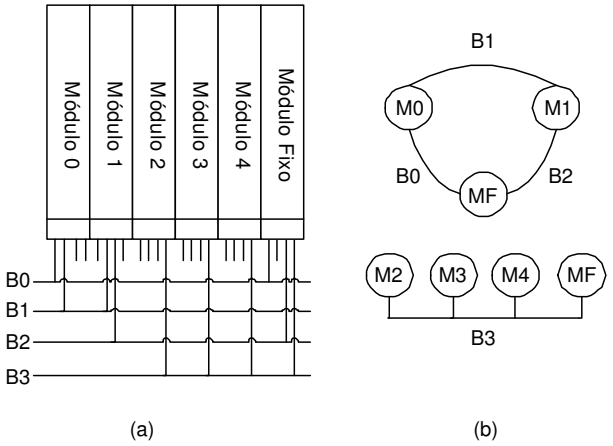


Figura 3.7: Exemplo de uma possível topologia de rede obtida com a infra-estrutura proposta por Huebner [19].

Os módulos MF, M0 e M1 estão conectados via barramento B0, B1 e B2, o que neste caso configura uma topologia em anel. Os módulos M2, M3, M4 e MF utilizam o barramento B3 caracterizando uma topologia em forma de barramento.

## 3.2 Propostas de Controladores de Configurações

### 3.2.1 Proposta de Ullmann et al.

Em [46] Ullmann et al. descrevem um gerenciador de tarefas de hardware para sistemas dinamicamente reconfiguráveis com restrições de tempo, com estrutura ilustrada na Figura 3.8. Este gerenciador de tarefas foi projetado para controlar a comunicação entre módulos funcionais da arquitetura e componentes externos (sensores e atuadores), através de um barramento específico. Esta arquitetura foi inicialmente projetada para aplicações automotivas tais como acionamento de vidros, controle do limpador de pára-brisas e controle do cinto de segurança.

A comunicação entre componentes externos e o controle do sistema ocorre por trocas de mensagens. As mensagens recebidas dos sensores são transformadas em um formato interno de representação, antes de serem enviadas para o módulo de controle da tarefa. O mesmo ocorre no sentido contrário, dos módulos de controle para os atuadores. O gerenciador é implementado em software e também controla o processo de reconfiguração dos módulos funcionais e o salvamento de seus contextos durante o tempo em que permanecem inativos no sistema.

O gerenciador de tempo real proposto em [46] é constituído por quatro partes principais,

conforme apresentado na Figura 3.8:

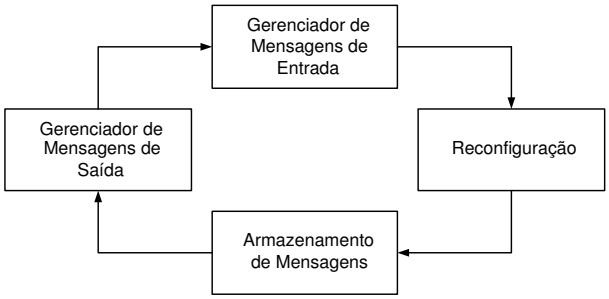


Figura 3.8: Gerenciador de tarefas de hardware para SDRs proposto por Ullmann et al.[46].

- *(i)Processo de Reconfiguração*: salva o estado do módulo funcional, reconfigura a arquitetura e restaura o estado de informações do novo módulo;
- *(ii)Gerenciador de Mensagens de Entrada*: recebe mensagens dos componentes externos, traduz e produz uma mensagem padronizada pelos módulos do sistema;
- *(iii)Gerenciamento de Mensagens de Saída*: recebe mensagens dos módulos, traduz e envia aos componentes externos do sistema.
- *(iv)Unidade de Gerenciamento de Armazenamento de Mensagens*: armazena e repassa mensagens que não puderam ser enviadas aos seus destinos devido a estes terem sido desativados e removidos pelo processo de reconfiguração.

### 3.2.2 Proposta de Resano et al.

Em [38] Resano et al. apresentam uma proposta de escalonamento de tarefas em SDRs. O objetivo da proposta é reduzir o custo temporal da reconfiguração de tarefas, habilitando o desenvolvimento de SDRs, que possuam múltiplas configurações parciais. Para isso, duas técnicas foram desenvolvidas. A *técnica de pré-busca*, onde a necessidade de uma tarefa é prevista e sua configuração pode ocorrer antes de sua possível utilização. A outra estratégia, a *técnica de reuso* de tarefas, permite que a penalidade de reconfiguração seja evitada, quando uma tarefa já residente no dispositivo é necessária. A proposta dos Autores consiste em aplicar tais estratégias e realizar uma parte do escalonamento em tempo de execução. Com isso, o tempo de reconfiguração pode ser reduzido; os possíveis erros de predição podem ser prevenidos e o atraso total de execução pode ser reduzido.

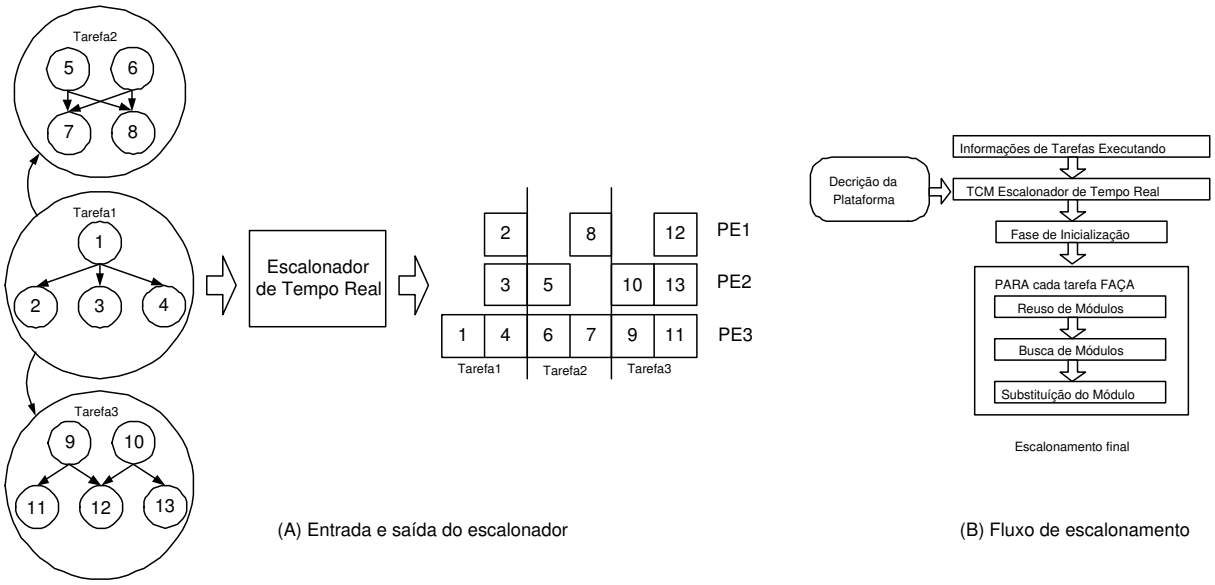


Figura 3.9: Técnicas de Resano et al..

Na Figura 3.9(a), apresenta-se um esboço das entradas e saídas do escalonador proposto por Resano et al. A aplicação é modelada como um conjunto de tarefas. Cada tarefa é composta por um conjunto de sub-tarefas, representadas na forma de um grafo. As tarefas podem reagir a estímulos externos, provocando a reorganização de suas sub-tarefas, gerando diferentes versões de grafos. Essas versões recebem o nome de *cenários*. A Figura 3.9(b) apresenta o fluxo de tarefas escalonamento proposto.

Em [39] e [37], Resano et al. afirmam que a operação de escalonamento pode se valer de técnicas mais apuradas e também mais lentas quando realizadas em tempo de projeto. Entretanto, estas não possuem informações suficientes sobre o estado do sistema durante sua execução. Por outro lado, o escalonamento em tempo de execução possui esse tipo de informação, mas deve ser rápido para obedecer às restrições temporais em SDRs. Por isso, não se deve empregar aqui estratégias muito complexas. Em virtude disso, Resano propõe um modelo de *escalonamento híbrido*, ou seja, parte realizado em tempo de projeto, parte em tempo de execução.

Em [37] os Autores detalham as estratégias de pré-busca e reuso. A Figura 3.10 apresenta um exemplo de resultados de um escalonamento, ilustrando a vantagem em aplicar pré-busca de tarefas, qual seja, ocultar o tempo de reconfiguração, pois esta é realizada antes da tarefa ser necessária. Em *(a)* apresenta-se a execução ideal do sistema, sem atraso de reconfiguração, apenas com o tempo de execução $Ex$ de cada tarefa. Em *(b)* o atraso de configuração $L$ de cada tarefa degrada o desempenho da aplicação. Em *(c)* o emprego de pré-busca reduz esse atraso.

Como se pode notar, apenas a configuração da primeira tarefa causa atraso de reconfiguração, enquanto que os demais atrasos foram sobrepostos pela execução da tarefa anterior. Uma pré-busca é realizada com base em uma *predição de reconfiguração*, uma analogia à predição de saltos.

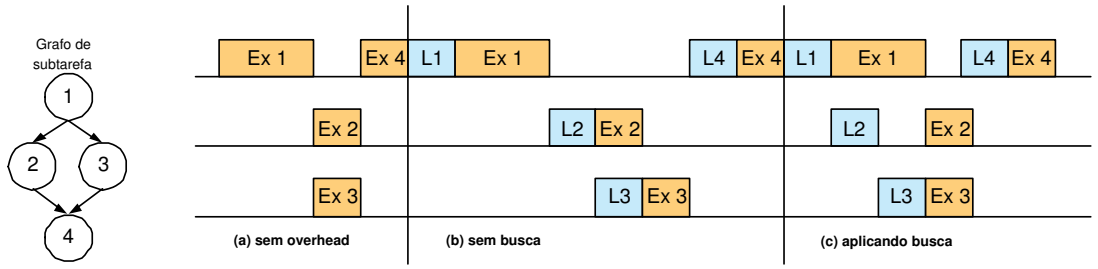


Figura 3.10: Exemplo de aplicação da técnica de pré-busca proposta por Resano et al..

Na Figura 3.11 a vantagem de realizar parte do escalonamento em tempo de execução é apresentada de uma forma gráfica. O reuso só pode ser aplicado em tempo de execução porque a identificação de tarefas reutilizáveis não pode ser realizada em tempo de projeto. O posicionamento das tarefas é realizado após o escalonamento. As tarefas são posicionadas em fatias virtuais, que serão associadas a fatias reais de acordo com a possibilidade de reuso. Pode-se notar, na Figura 3.10, que a tarefa 3 foi posicionada na fatia virtual 3. Como essa tarefa já se encontrava posicionada na fatia 1 (ver estado inicial do FPGA), então sua posição virtual foi mapeada para tal fatia. O mesmo acontece com as demais tarefas.

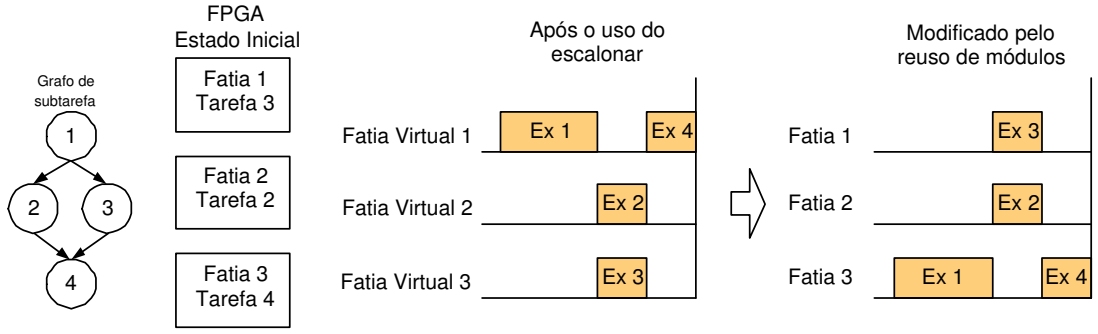


Figura 3.11: Exemplo de aplicação da técnica de pré-busca proposta por Resano et al..

### 3.2.3 Proposta de Mignolet et al.

Em [28], Mignolet et al. apresentam uma infra-estrutura para projeto e gerenciamento de tarefas realocáveis. No sistema proposto, as tarefas podem ter seu contexto de operação trocado

de forma que uma tarefa executando em hardware pode ser interrompida, e continuar seu processamento em software, e vice-versa. Dada a natureza do sistema alvo, existe a necessidade de uma interface de comunicação unificada. A comunicação entre as tarefas é realizada através de troca de mensagens. Cada tarefa possui um endereço específico. A comunicação entre duas tarefas pode ocorrer de três formas distintas, de acordo com o domínio no qual as tarefas encontram-se residentes, segundo a Figura 3.12. Quando ambas as tarefas estão executando em hardware, uma NoC é utilizada para comunicação. Quando as tarefas estão executando em software, no processador embarcado, a comunicação é realizada através de uma *API* (do inglês, Application Programming Interface). O terceiro tipo de comunicação acontece quando uma tarefa envolvida na comunicação está executando em software e a outra em hardware. Nesse caso, uma *camada de abstração de hardware* (em inglês, *Hardware Abstraction Layer - HAL*) é utilizada.

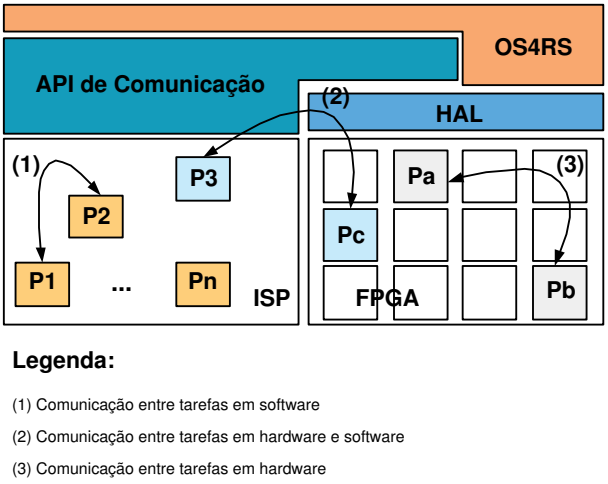


Figura 3.12: Infra-estrutura de hardware/software que permite a comunicação entre tarefas, tanto em hardware quanto em software [28].

### 3.2.4 Proposta de Griese et al.

Em [14], Griese et al. propõem a construção de um gerenciador de reconfiguração dinâmica (em inglês *Run-Time Reconfiguration Manager - RTR Manager*). O gerenciador foi implementado em hardware e tem a função de controlar, monitorar e executar o processo de reconfiguração em uma plataforma alvo. O sistema contém módulos reconfiguráveis implementados em hardware, um processador e barramentos de comunicação. Este sistema mantém armazenada as configurações parciais em um hospedeiro, com o qual se comunica através de um barramento

com interface PCI . O processador executa um sistema operacional de tempo real para aplicações com restrições de tempo. O gerenciador foi implementado com chaveamento de contexto e mecanismos de segurança para prevenir eventuais falhas durante o processo de reconfiguração. O sistema proposto utiliza a interface *SelectMap* para a reconfiguração de dispositivos FPGA da Xilinx. A implementação e teste do sistema foram realizados na plataforma de prototipação *Raptor2000* do Instituto *Heinz Nixdorf Institute - Padderborn* [23]. Esta plataforma é subdividida em módulos de expansão. A Figura 3.13 apresenta um diagrama de blocos do sistema implementado nesta plataforma.

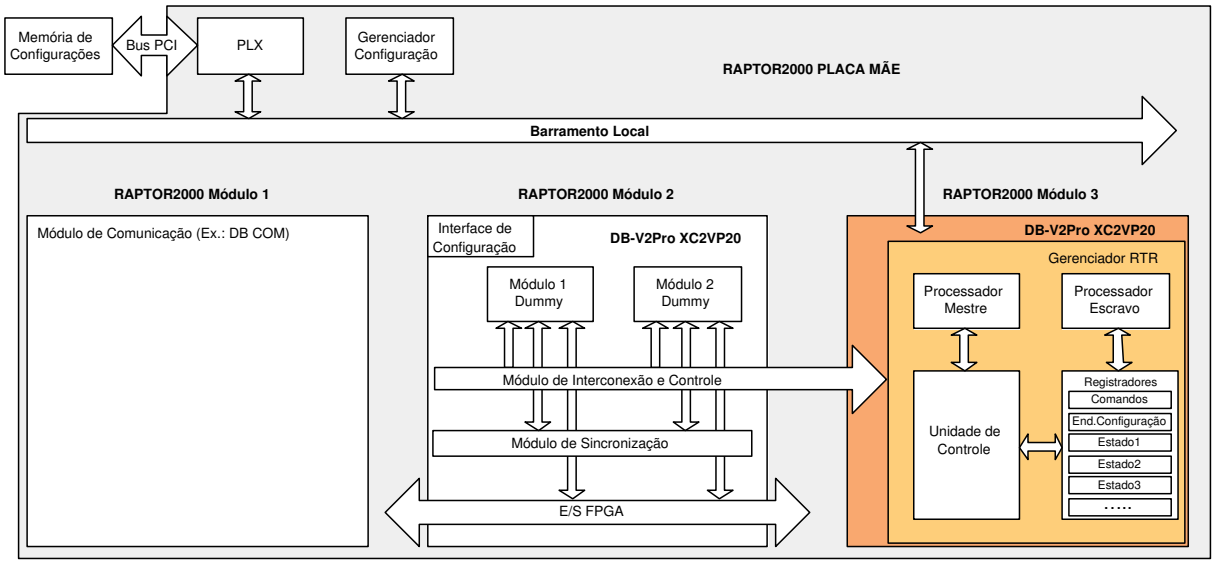


Figura 3.13: Diagrama de blocos da plataforma proposta por Griese[14].

### 3.2.5 Proposta de Williams e Bergmann

Em [47], Williams e Bergmann propõem a construção de um *driver de dispositivo* para o sistema operacional $\mu$Clinux embarcado em uma plataforma auto-reconfigurável. Esta plataforma é baseada no processador Microblaze, capaz de executar este kernel do Linux. O *driver de dispositivo* disponibiliza serviços a aplicações embarcadas, tais como: envio de bitstreams para a porta ICAP e compactação de bitstreams. Deste modo, o $\mu$Clinux abstrai as complexidades envolvidas no processo de reconfiguração e disponibiliza comandos para serem executados no Shell do sistema. Com isso, os autores visam utilizar uma plataforma com um sistema operacional flexível e de código aberto, adaptando as ferramentas Linux para auxiliar na resolução de problemas encontrados em sistemas auto-reconfiguráveis.

### 3.2.6 Proposta de Carvalho et al.

Em [7], Carvalho et al. propõem um subsistema que controle a dinâmica de execução dos módulos funcionais em sistemas reconfiguráveis. A tarefa de controlar dinamicamente a execução de módulos em um dispositivo reconfigurável deve ser atribuída a uma entidade. Carvalho propôs o *Gerenciador de Configuração de Sistemas Reconfiguráveis - RSCM* (do inglês, *Reconfigurable System Configuration Manager*), um modelo de controlador de reconfiguração capaz de desempenhar funções semelhantes às de um sistema operacional. A estrutura do RSCM implementada integralmente em hardware é apresentada na Figura 3.14 [7]:

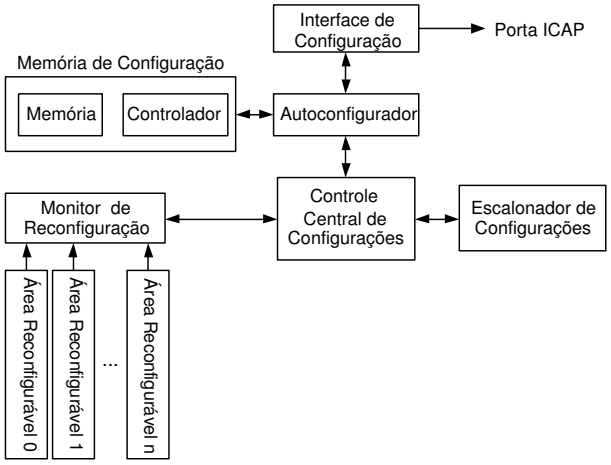


Figura 3.14: Modelo de um RSCM para implementações de SDRs, retirada de [7].

Os principais módulos que compõem o controlador RSCM e suas funções estão descritas a seguir:

*(i)Memória de Configuração(CM)*, armazena todos bitstreams parciais usados pelo sistema em tempo de execução. Levando em conta o escasso espaço de armazenamento em FPGAs;

*(ii)Autoconfigurador(SC)*, controla o processo físico de configuração/reconfiguração;

*(iii)Interface de Configuração(CI)* formata os dados de configuração para o FPGA.

*(iv)Controle Central de Configuração(CCC)*, recebe requisições para iniciar o processo de configuração e fornece resultados do processo na forma de sinais de status.

*(v)Monitor de Reconfiguração(RM)*, detecta as situações onde uma reconfiguração pode/deve ser realizada.

*(vi)Módulo Escalonador(CS)*, é responsável por determinar qual bitstream é o próximo a ser configurado. Este módulo recebe requisições do CCC e armazena uma estrutura de dados com informação sobre dependências entre configurações, como uma tabela. Esta tabela é chamada

*Tabela de Dependências e Descritores (TDD).*

## 3.3 Discussão das Propostas Revisadas

Entre os trabalhos revisados e discutidos na Seção 3.2, destacam-se alguns que servem de referencial teórico para este trabalho. Nesta Seção são discutidos aspectos positivos e negativos das propostas.

O gerenciador de configurações proposto por Ullmann é dedicado a aplicações específicas, como unidades de controle de funções automotivas. Estas aplicações são classificadas como *reativas*, ou seja, o sistema toma decisões um função das ações detectadas por sensores que compõe o sistema. O gerenciador atende a tarefas em tempo real com restrições não críticas, ou seja, tarefas que não sejam vitais ao controle do automóvel, como o acionamento do limpador de vidros. O gerenciador não possui uma unidade de escalonamento de tarefas, mas estabelece níveis de prioridades às tarefas, favorecendo algumas tarefas segundo uma política definida em tempo de projeto.

Resano et al. em [38],[39], [37] propõem o uso de técnicas para o escalonamento de tarefas em SDRs. Estas técnicas, conforme descrito anteriormente, buscam esconder da aplicação o tempo gasto no processo de reconfiguração do hardware. Estas técnicas implementadas em software são úteis no projeto e desenvolvimento de controladores de configuração para aplicações embarcadas em plataformas com duas ou mais áreas reconfiguráveis.

Mignolet et al. em [28] propõem uma infra-estrutura em hardware e software para suporte a um sistema operacional para tarefas de tempo real implementado em software. Esta estrutura permite que durante a execução do sistema, tarefas implementadas em software sejam realocadas em hardware para que consigam atender às restrições de tempo impostas pelo sistema. O co-projeto de hardware e software permite a implementação de uma dada tarefa tanto em software quanto em hardware. Deste modo, durante a execução, o sistema operacional emprega chaveamento de contexto, permitindo que uma dada tarefa seja realocada e mantenha seu contexto de execução. Nesta proposta, a reconfiguração parcial do sistema tem a função de configurar tarefas em hardware ou a desativação da tarefa caso esta seja realocada em software. O projetista define em tempo de projeto a política de escalonamento das tarefas de acordo com os tempos de trocas de contextos e de execução de tarefas. Esta proposta foi implementada sobre o dispositivo XC2V6000 da família VirtexII do Xilinx.

não existe referência ao dispositivo sobre o qual se realiza a configuração parcial.

Griese et al. em [14] propõem uma infra-estrutura para projetos de SDRs. Esta infra-estrutura contém um controlador de configurações implementado em hardware e um processador PowerPC, que juntos suportam um sistema operacional para aplicações de tempo real. A arquitetura de hardware proposta é embarcada em uma plataforma de prototipação específica chamada *Raptor2000*. Este sistema emprega barramentos próprios da plataforma para realizar a reconfiguração. O controlador de configurações proposto disponibiliza funções como chaveamento de contexto e mecanismos de segurança para prevenir eventuais falhas durante o processo de reconfiguração do sistema, conforme descrito anteriormente. Esta proposta apesar de usar dispositivos da família VirtexII-Pro da Xilinx, a qual também emprega-se neste trabalho, utiliza recursos específicos da plataforma. Além disso, utiliza um hospedeiro para armazenar as configurações parciais do sistema.

O trabalho proposto por Carvalho [7], como discutido em [44], apresenta uma restrição importante, o não funcionamento da porta interna de acesso a configuração (ICAP) disponível em FPGAs da família Virtex da Xilinx. Esta limitação impede que este controlador de configurações, totalmente desenvolvido em hardware, dê suporte a aplicações auto-reconfiguráveis. No presente trabalho dominou-se o controle e funcionamento da porta ICAP, disponibilizando uma interface de acesso a aplicações em software.

A Tabela 3.1 apresenta uma comparação entre os trabalhos estudados de maneira a destacar características relevantes à proposta deste trabalho.

Tabela 3.1: Comparação entre as propostas de controladores.

| **Atividades** | **Tipo de Implementação** | **Uso em Sistema Operacional** | **Interface de Reconfiguração** | **Chaveamento de Contexto** |
|---|---|---|---|---|
| Ullmann [46] | Sw | Não | ICAP | Sim |
| Resano [38] | Sw | - | - | Sim |
| Mignolet [28] | Sw/Hw | Sim | - | Sim |
| Griese [14] | Hw | Sim | Select Map | Sim |
| Williams [47] | Sw | Sim | ICAP | - |
| Carvalho [7] | Hw | Não | Select Map | Não |
| RSCM-S | Sw | Não | ICAP | Não |

# Capítulo 4

# Ferramentas e Fluxo de Projeto

O desenvolvimento de SDRs requer a utilização de ferramentas que ofereçam suporte necessário ao projeto da infra-estrutura de hardware/software. A carência destas ferramentas e a dependência de uma dada tecnologia, conforme indicado no Capítulo 2, motivam a pesquisa e desenvolvimento de ferramentas e fluxos de projeto alternativos aos atuais propostos até mesmo por fabricantes que dominam a tecnologia.

O domínio do mercado de dispositivos que suportam reconfiguração dinâmica e parcial por um fabricante aumenta a dependência por esta tecnologia. Isto faz com que os métodos de projeto utilizados sejam exclusivos para esta tecnologia, restringindo sua expansão para plataformas de outros fabricantes.

Neste Capítulo são apresentadas as ferramentas de desenvolvimento utilizadas no projeto de infra-estrutura de SDRs. Segundo a opção realizada no Capítulo 2, serão abordados apenas os ambientes de desenvolvimento da Xilinx, empresa que domina o mercado.

O restante deste Capítulo é organizado da seguinte maneira. A Seção 4.1 apresenta o ambiente EDK e a estrutura base de arquiteturas microprocessadas por ele suportadas. A Seção 4.2 apresenta o suporte necessário ao projeto de uma arquitetura auto-reconfigurável. A Seção 4.3 discute aspectos da tecnologia suprida pela Xilinx para dar suporte a sistemas auto-reconfiguráveis.

## 4.1 Ambiente de Desenvolvimento EDK para Sistemas Embarcados

A Xilinx comercializa um ambiente de desenvolvimento com o objetivo de agrupar ferramentas de projeto que permitam ao projetista construir sistemas computacionais microprocessados

complexos. Este ambiente é conhecido como *Kit de Desenvolvimento de Sistemas Embarcados*(em inglês, *Embedded Development Kit, ou EDK*). O ambiente é basicamente formado por um repositório de componentes, associado a um conjunto de ferramentas de projeto com os quais o projetista pode interagir através de uma interface gráfica, definindo a estrutura do sistema. A saída do ambiente é um arquivo de configuração para algum FPGA Xilinx. Uma visão simplificada do ambiente de desenvolvimento aparece na Figura 4.1.

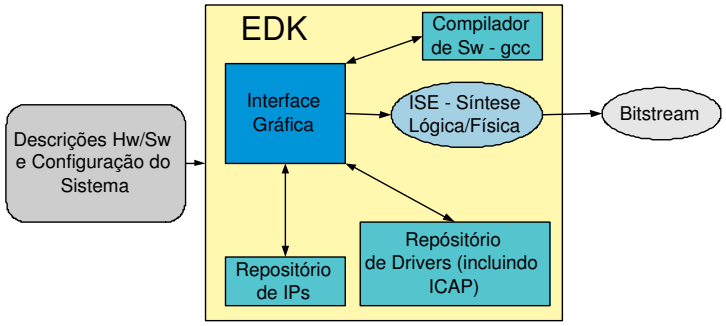


Figura 4.1: Uma visão geral do ambiente EDK: repositórios e ferramentas para desenvolvimento de projetos de hardware e software.

As ferramentas disponíveis no ambiente EDK suportam tanto o projeto de hardware quanto o projeto de software para sistemas computacionais. Além disso, opcionalmente, o ambiente suporta verificação e simulação dos sistemas projetados.

O ambiente EDK também utiliza ferramentas de projeto de outro ambiente oferecido pela Xilinx, o ambiente *ISE* (em inglês, *Integrated Software Environment*). As ferramentas do ISE complementam o fluxo de projeto de hardware do EDK. Para o desenvolvimento de projetos de software embarcado, o EDK oferece uma plataforma suporte para projetos em *linguagem C/C++* capaz de gerar código executável para os processadores disponíveis.

O EDK possui um repositório de núcleos IP de hardware que podem ser utilizados pelo projetista, além de permitir a criação e agregação de periféricos ao sistema embarcado com lógica definida pelo próprio projetista. Este ambiente emprega uma estrutura própria para a criação de arquiteturas microprocessadas. Dentre os principais componentes desta estrutura, citam-se os seguintes:

- Microprocessadores
- Arquitetura de Comunicação

- Memórias
- Repositório de IPs
- Fluxo de Criação de IPs
- Plataforma de Desenvolvimento de Software

A partir destes componentes, é possível construir sistemas computacionais para diversas aplicações conforme descrito, por exemplo, em [46], [18]. A Figura 4.2 ilustra uma arquitetura de um sistema digital composto de processador e hardware dedicado, contando com um mínimo de recursos de hardware e software, conforme obtido pelo emprego dos ambientes EDK e ISE.

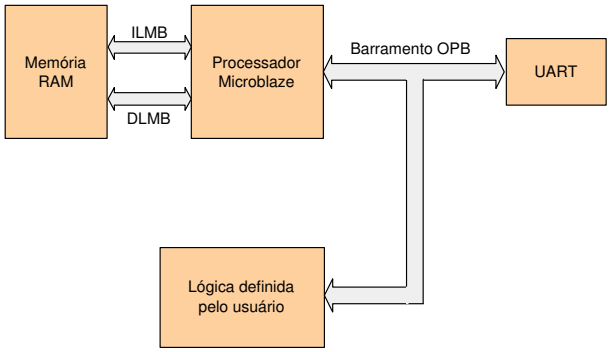


Figura 4.2: Arquitetura mínima de um sistema digital composto de hardware e software dedicado, construída através do EDK.

### 4.1.1 Microprocessadores

O ambiente EDK suporta dois microprocessadores distintos para a criação de sistemas embarcados. São eles o MicroBlaze e o PowerPC405. O microprocessador MicroBlaze está disponível no EDK como um núcleo IP "firm", ou seja, uma descrição pré-sintetizada, a qual o ambiente não permite edição manual. Este processador apresenta as seguintes características:

- barramento de dados com largura de 32 bits;
- arquitetura load/store, pipeline com 3 estágios, 32 registradores de propósito geral (32 bits), suporte à memória cache. (RISC) (em inglês, Reduced Instruction Set Computer );
- arquitetura *Harvard* (utiliza separadamente barramentos para dados e instruções);

- interface de comunicação para o barramento *OPB* (em inglês, On-chip Peripheral Bus) que será discutida na Seção 4.1.2.

O processador PowerPC405 é um processador desenvolvido pela empresa IBM e encontra-se disponível no ambiente como um núcleo IP "hard". Este processador é encontrado em FPGAs das famílias VirtexII-Pro e VirtexIV FX. O PowerPC405 apresenta as seguintes características:

- barramento de dados com largura de 32 bits;
- arquitetura load/store (RISC), pipeline com 5 estágios, 32 registradores de propósito geral (32 bits), suporte à memória cache;
- arquitetura *Harvard*;
- interface de comunicação para o barramento *PLB* (em inglês, *Processor Local Bus*).

A Xilinx disponibiliza uma série de notas de aplicação (em inglês, *application notes*) com descrições de projetos desenvolvidos através do ambiente EDK. Por exemplo, a nota *XAPP 662* [57] apresenta uma aplicação exemplo desenvolvida para detecção de erros em comunicação serial, chamada BERT (em inglês, *Bit Error Rate Tester*)[59], [56]. Esta aplicação utiliza o processador PowerPC405, fato que torna o sistema dependente das famílias VirtexII-Pro e VirtexIV VFX. Para o desenvolvimento da infra-estrutura alvo optou-se por usar o processador MicroBlaze, que possibilita a prototipação sobre todos os FPGAs listados no Capítulo 2.

### 4.1.2 Arquitetura de Comunicação

O ambiente EDK utiliza uma arquitetura de barramento responsável por estabelecer um meio de comunicação entre o microprocessador e periféricos. Esta arquitetura desenvolvida pela IBM é denominada *CoreConnect*. Ela é disponibilizada em três diferentes interfaces de comunicação: interfaces OPB (*On-chip Peripheral Bus*), PLB (*Processor Local Bus*) e DCR (*Device Control Register Bus*) . Informações detalhadas sobre estes barramentos oferecidos no EDK podem ser obtidas respectivamente em [21], [22] e [20].

### 4.1.3 Repositório de IPs

O ambiente EDK contém um repositório de núcleos IPs disponíveis para uso no projeto de sistemas embarcados. Para cada núcleo IP, o ambiente disponibiliza também drivers para

controle destes IP em software. O ambiente permite também o desenvolvimento de núcleos IPs com lógica definida pelo projetista capazes de se comunicar com o restante do sistema através do barramento CoreConnect. Os detalhes sobre a criação de núcleos IPs no ambiente EDK podem ser obtidos através dos manuais da Xilinx, tal como [54].

### 4.1.4 Memórias

Qualquer sistema microprocessado precisa obrigatoriamente de memória para armazenar o programa a ser executado pelo sistema. O ambiente EDK dispõe em seu repositório de IPs controladores de memória para o projeto de sistemas embarcados. O tipo de memória empregada em um dado sistema depende exclusivamente da plataforma de prototipação escolhida. Tradicionalmente, são encontradas plataformas de prototipação com os seguintes tipos de memórias: SRAMs, SDRAMs e DDR SDRAMs. Uma outra opção é usar blocos de memória internos ao FPGA as BRAMs.

BRAMs (em inglês, *Block Random Access Memory*) são internas aos FPGAs das famílias Virtex. Sua capacidade de armazenamento é pequena comparada às memórias externas. Como exemplo, a plataforma VirtexII-Pro$^{TM}$FF1152 da Memec Insight, contém 306KB de BRAMs no seu FPGA e 64MB em SDRAM externa.

As SRAMs (em inglês, Static RAMs) são memórias de alto custo e desempenho. Portanto, encontram-se em quantidades pequenas, ou seja, na ordem de alguns MBytes por plataforma. A plataforma utilizada não contém memória SRAM, apenas a placa de expansão *P160* contém este tipo de memória.

As SDRAMs (Synchronous Dynamic RAMs) encontram-se disponíveis na maioria das plataformas e são de grande porte. A plataforma utilizada neste trabalho contém 64 MB de SDRAM, conforme descrito anteriormente. As memórias SDRAM DDRs (em inglês, SDRAMs Double Data Rate) são memórias mais rápidas e encontradas em plataformas mais recentes, como exemplo as plataformas da Memec-Insight com VirtexIV embarcados. Estas memórias também apresentam-se em configurações de grandes capacidade. As plataformas costumam possuir *slots* de expansão para pentes de memória, muitas vezes compatíveis com os utilizados em computadores pessoais.

### 4.1.5 Fluxo de Criação de IPs

O ambiente EDK permite ao projetista criar núcleos IPs capazes de se comunicar com o processador da arquitetura. Esta facilidade oferecida pelo ambiente EDK permite aos projetistas a criação de arquiteturas especializadas, de forma a atender melhor os requisitos de um dado projeto. O fluxo de projeto para criação de IPs gera esqueletos (em inglês, *templates*[1]) com a estrutura de hardware necessária para que seja estabelecida a comunicação entre IP e processador através do barramento CoreConnect.

No ambiente EDK, a comunicação entre o IP criado e a arquitetura é realizada através da interface *IPIF* (em inglês, *Intellectual Property Interface*) permitindo que IPs não necessitem comunicar-se diretamente ao barramento CoreConnect. A interface IPIF disponibiliza um subconjunto de sinais que estabelecem a comunicação física entre IPs. Estes sinais são denominados de interconexão dos IPs, *IPIC* (em inglês, *Intellectual Property InterConnect*). A Figura 4.3 representa um núcleo IP criado pelo usuário com sua interface de comunicação.

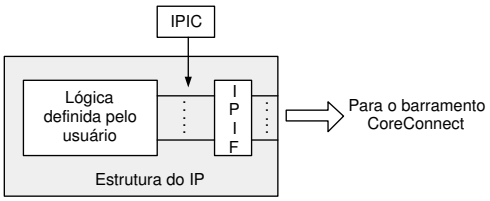


Figura 4.3: Estrutura do template de hardware gerado através do fluxo de criação de IPs do EDK.

No projeto de SDRs, os IPs que ocupam áreas reconfiguráveis da arquitetura são projetados através do fluxo de criação de IPs do usuário do EDK. Como visto na Figura 4.3, a interface de comunicação entre áreas fixa e reconfiguráveis é definida pelo conjunto de sinais IPIC da interface IPIF. Para que esta interface de comunicação tenha uma posição fixa na arquitetura são utilizadas as macros discutidas no Capítulo 3. Estas macros são inseridas entre o módulo IPIF e o barramento Coreconnect. Detalhes sobre macros e seu uso são descritos na Seção 4.2.2.1.

### 4.1.6 Plataforma de Desenvolvimento de Software

O EDK oferece suporte ao projeto de software executado sobre a arquitetura de hardware por ele projetado. A plataforma de desenvolvimento contida no EDK utiliza compiladores da linguagem C/C++ para ambos processadores, Microblaze e PowerPC405.

[1] Template utilizado neste contexto vem a ser uma descrição de hardware genérica parametrizável.

O fluxo de projeto para criação de arquiteturas de hardware gera o arquivo chamado *MSS* (em inglês, *Microprocessor Software Specification*). Este arquivo relaciona os drivers necessários para o controle dos IPs pertencentes à arquitetura do sistema projetado. Este arquivo é usado como entrada para a ferramenta *LibGen*, cuja função é gerar bibliotecas com drivers necessários aos IPs da aplicação.

Após a geração de bibliotecas, é realizada a compilação do código fonte da aplicação e a ligação com os drivers, para que seja obtido o código executável para o processador da arquitetura. Este código deve ser alocado em uma região específica de memória no momento da prototipação do sistema, para que o processador realize a carga do sistema de software e execute a aplicação projetada.

O EDK oferece fluxos concomitantes de hardware e software, utilizando diferentes ferramentas de desenvolvimento. Para que ao final do projeto seja obtido um arquivo de configurações contendo ambos projetos, o EDK oferece uma ferramenta denominada *Data2Mem* com a função de unir ambos projetos para realizar a configuração do FPGA.

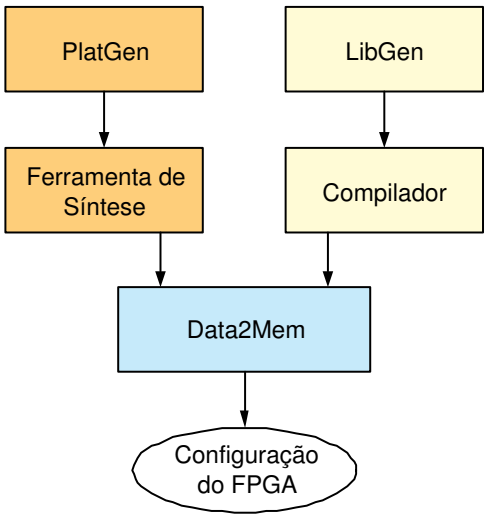


Figura 4.4: Representação dos fluxos de projetos de hardware e software unidos pela ferramenta Data2Mem para que ocorra a prototipação do sistema.

### 4.1.7 Fluxo de Síntese dos Ambientes EDK e ISE

O projeto de hardware em plataformas de prototipação utiliza ferramentas que obtêm como entrada a descrição de uma arquitetura e produzem código para configurar a memória de controle de um FPGA, para que este tenha o comportamento desejado pelo projetista. Nesta Seção são

apresentadas as etapas que transformam estas descrições em configurações para dispositivos FPGAs, etapas denominadas de fluxo de síntese lógica e síntese física, conforme oferecidas pelos ambientes EDK e ISE.

A Figura 4.5 apresenta a descrição das etapas para realização de síntese, simulação e verificação de projetos em hardware.

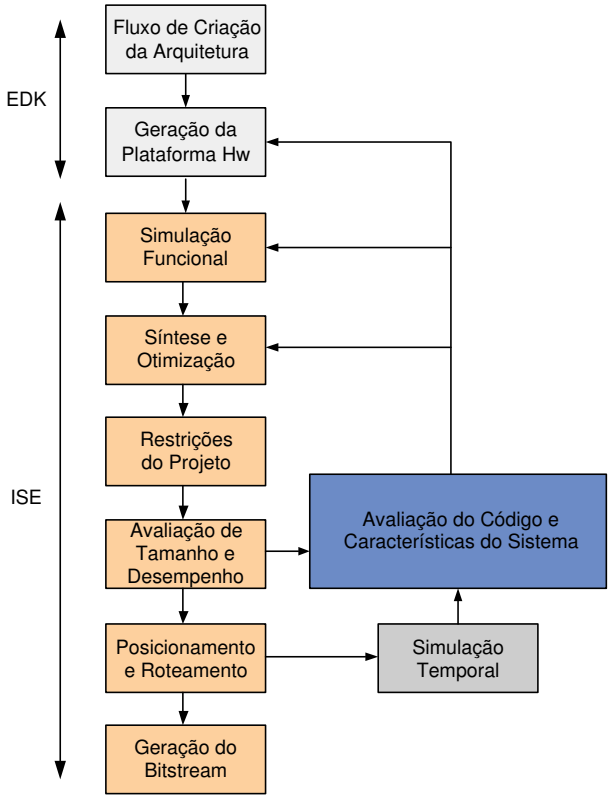


Figura 4.5: Fluxo de síntese lógica e física realizado pelos ambientes EDK e ISE.

**Criação da Arquitetura:** A primeira etapa do fluxo é a descrição do projeto de hardware. A interface gráfica do EDK, denominada XPS (em inglês, *Xilinx Platform Studio*), possui um fluxo de criação de arquiteturas que permite ao projetista selecionar IPs que irão compor a arquitetura, permitindo ainda parametrizá-los. Esta etapa gera uma descrição dos IPs selecionados, denominada como *especificação do hardware do processador* (em inglês, Microprocessor Hardware Specification - MHS) .

**Geração de Plataforma:** Nesta etapa, a ferramenta *Platgen* realiza a geração da arquitetura de hardware em formato *netlist* a partir do MHS e dos IPs. O formato netlist é uma descrição da arquitetura de hardware em nível de portas e suas interconexões. Platgen gera uma hierarquia de arquivos de acordo com as conexões de IPs que constituem a arquitetura do

sistema. Estas interligações constituem a plataforma de hardware em formato *netlist*.

**Simulação Funcional:** O EDK oferece uma interface para a simulação de sistemas computacionais, que juntamente com um ambiente de simulação, tal como o *ModelSim*, oferecem três níveis de simulações: comportamental, funcional e temporal. Nesta etapa do fluxo, é realizada a simulação comportamental, ou seja, uma simulação do comportamento do sistema antes da síntese propriamente dita do hardware. Esta etapa é um ponto de verificação para que problemas de implementação sejam detectados nas fases iniciais de projeto.

**Síntese e Otimização:** Após a geração do netlist do sistema, o fluxo de síntese passa a ser desenvolvido no ambiente ISE. O EDK utiliza as ferramentas do ISE para realizar a síntese do hardware, mas não permite que o projetista tenha acesso direto a estas ferramentas. O EDK oferece um *prompt* para que o usuário use as ferramentas do ISE através de uma interface de linha de comando. O ambiente ISE disponibiliza acesso a várias ferramentas para síntese física, tais como o *Leonardo Spectrum* e o *XST*, este último do próprio fabricante.

**Restrições de Projeto:** Esta etapa permite ao usuário impor restrições ao processo de síntese da arquitetura, como por exemplo definir as coordenadas da posição ocupada por IPs sobre a matriz de CLBs do FPGA, e definir pinos do dispositivo para entrada e saída de dados da arquitetura. Este é um passo importante no projeto de SDRs, porque permite o posicionamento das interfaces entre áreas fixas (onde reside o controle do sistema) e reconfiguráveis. Os ambientes EDK e ISE armazenam estas informações em um arquivo de restrições do usuário/projetista denominado <nome_projeto>.UCF (em inglês, *User Constraint File*).

**Avaliação de Tamanho e Desempenho:** Após a etapa de síntese, o projeto de hardware passa a ser descrito como uma configuração dos recursos existentes no FPGA. Desta forma, é possível analisar o tamanho ocupado pelo projeto e desempenho do mesmo, de acordo com as restrições realizadas na etapa anterior. Nesta etapa, ainda não é possível realizar uma verificação temporal do sistema, pois para isso é necessário definir as coordenadas de CLBs e fios que irão efetivamente compor a arquitetura do sistema. Após realizar a avaliação do sistema, é possível retornar a etapas anteriores para que sejam realizadas alterações, de modo que o sistema cumpra às restrições de projeto.

**Posicionamento e Roteamento:** Nesta etapa, a ferramenta de síntese define quais recursos específicos do FPGA serão usados para implementar a arquitetura, de acordo com as restrições impostas anteriormente. A ferramenta realiza o roteamento de fios para a interconexão dos IPs da arquitetura. Este roteamento é realizado segundo um algoritmo definido pelo fabricante e não

permite que o usuário interfira neste processo. A determinação de coordenadas para cada CLB também não é possível, mas a ferramenta permite que regiões de CLBs sejam determinadas para conter cada IP da arquitetura.

Ao final desta etapa, a ferramenta gera um arquivo contendo o resultado do posicionamento e do roteamento para a arquitetura. Este arquivo é chamado *<nome_projeto>.NCD* (do inglês, *Native Circuit Description*) e contém informações sobre todos os recursos do FPGA utilizados para implementar a arquitetura. A partir deste arquivo é possível realizar uma verificação visual da arquitetura sintetizada sobre os recursos do FPGA através da ferramenta *FPGA Editor*. O FPGA Editor permite alterar manualmente algumas definições como, por exemplo a função lógica de LUTs e o roteamento individual de fios.

O posicionamento e roteamento também geram relatórios sobre a análise temporal e quantitativa realizada durante o processo de síntese, permitindo ao projetista verificar se a arquitetura atende aos requisitos temporais de projeto e verificar se o tamanho da arquitetura confere com a análise quantitativa realizada durante a síntese. O relatório sobre a quantidade de recursos utilizados para implementação do sistema permite a análise quantitativa do sistema medida em Slices, LUTs e flip-flops tipo D.

**Geração do Bitstream:** A última etapa do processo de síntese gera o arquivo de configuração para o FPGA. A ferramenta *Bitgen* gera bitstreams a partir das descrições contidas no arquivo *.NCD*. A saída padrão desta ferramenta é um arquivo binário com a configuração do FPGA. Opcionalmente a ferramenta oferece a geração de um arquivo ASCII que representa através de caracteres a mesma informação do formato binário. Isto permite a análise do arquivo de configurações e por conseqüência a construção da ferramenta para geração de bitstreams parciais tal como aquela proposta e desenvolvida por Möller [29].

As carência de ferramentas para o projeto de SDRs e as restrições encontradas nas ferramentas existentes motivam a construção de alternativas para o desenvolvimento de SDRs encontradas neste trabalho. A partir deste fluxo de síntese lógica e física desenvolvido pela Xilinx através dos ambientes EDK e ISE é possível desenvolver um novo fluxo para geração de sistemas dinâmica e parcialmente reconfiguráveis. A Seção 4.2 apresenta uma proposta de fluxo de desenvolvimento de SDRs, que constitui uma contribuição deste trabalho.

## 4.2 Utilização do ambiente EDK para o desenvolvimento de SDRs

Um novo fluxo de projeto desenvolvido para implementar SDRs surgiu como uma alternativa aos fluxos propostos pela Xilinx em [60]. Esta proposta de concepção de SDRs também usa ferramentas dos ambientes EDK e ISE, juntamente com uma ferramenta desenvolvida por Möller em [29] para análise e geração de bitstreams parciais, denominada *CoreUnifier-II Pro*.

O fluxo de desenvolvimento pode ser dividido em 4 etapas de projeto bem definidas: Concepção da Arquitetura, Definição de Planta Baixa e Síntese Lógica e Física, Geração de Bitstreams Parciais e Projeto de Software. A Figura 4.6 apresenta uma visão do fluxo e iterações existentes entre as etapas.

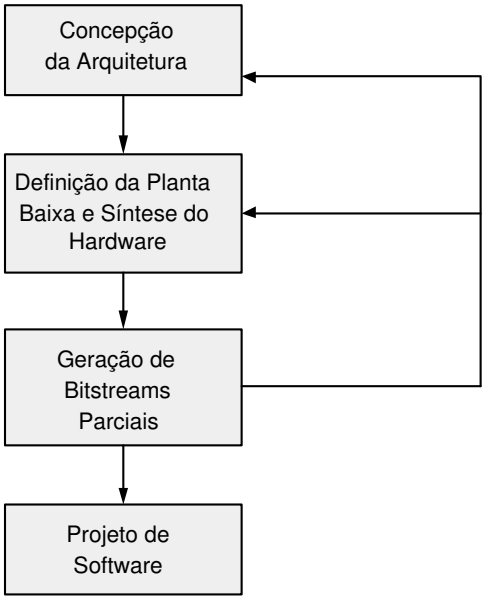


Figura 4.6: Uma visão geral do fluxo de projeto e suas interações.

### 4.2.1 Concepção da Arquitetura

Esta é a primeira etapa do fluxo de projeto. Nesta etapa são definidos os núcleos IPs que irão compor a *arquitetura base* do sistema. A *arquitetura base* é a configuração total do FPGA que contém toda a parte fixa do sistema e uma configuração inicial para cada uma das áreas reconfiguráveis. Através do fluxo de criação de arquiteturas do EDK, constrói-se a arquitetura base, onde são definidos: o processador, os barramentos, periféricos para entrada e saída de dados e tipos de memória. Nesta etapa, também é definida a criação e adição de novos periféricos à arquitetura base.

A cada periférico reconfigurável criado pelo projetista deve-se agregar macros para formar a interface (fixa) entre uma área fixa e uma reconfigurável ou entre duas áreas reconfiguráveis. Macros são inseridas entre o barramento CoreConnect e a interface IPIF do periférico, pois este é o único modo que permite o posicionamento das macros com a ferramenta Floorplanner. A Figura 4.7 mostra a posição da macro dentro da estrutura do periférico.

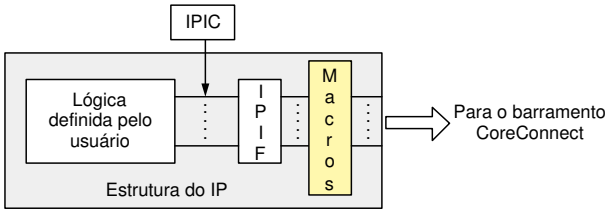


Figura 4.7: Inserção das macros no periférico criado pelo usuário, formando a interface de comunicação entre áreas fixa e reconfigurável ou entre duas áreas reconfiguráveis.

A área fixa definida como a área de controle de um SDR, por exemplo é composta pelo processador MicroBlaze e todos os periféricos necessários ao seu processamento como: memórias, barramentos, controladores para dispositivo ICAP e interrupção, e ainda dispositivos de entrada e saída. As áreas reconfiguráveis são representadas por IPs definidos pelo usuário para desempenhar uma tarefa específica no sistema.

Após a definição dos IPs que compõem a arquitetura base do sistema com a introdução das macros, deve ser gerado o *netlist* da arquitetura, ou seja, é dado início ao processo de síntese e otimização do hardware, conforme a descrição do fluxo de síntese apresentado na Seção 4.1.7.

### 4.2.2 Definição da Planta Baixa e Síntese Lógica e Física

O ambiente ISE disponibiliza as ferramentas necessárias para esta etapa do projeto. Emprega-se a ferramenta *Floorplanner* para definir a planta baixa do sistema, ou seja, estabelecer a dimensão (formato e medidas) de cada área que conterá cada módulo de hardware da arquitetura dentro do FPGA. O uso desta ferramenta deve atender a restrições impostas pelo projeto e pelo hardware externo, incluindo: localização de pinos de entrada e saída do dispositivo, proximidade entre os módulos e isolamento entre áreas da arquitetura. Para isolar as áreas é necessário o uso das macros que fixam a interface de comunicação entre as áreas. Segundo estudos realizados no Capítulo 3, existem propostas de criação de interfaces entre áreas. Porém, neste trabalho será empregado um novo modelo de interface, definido por Möller [29]. Möller desenvolveu *Macros* que possibilitam a comunicação e o isolamento entre IPs durante a etapa de concepção

da arquitetura.

A definição de áreas depende, entre outros fatores das restrições impostas pela arquitetura de (re)configuração do dispositivo em questão. Uma maneira de definir áreas é dada a seguir, voltada para criar sistemas a serem implementados sobre os dispositivos VirtexII e VirtexII-Pro da Xilinx. Nestes dispositivos a ICAP encontra-se localizada no canto inferior direito do FPGA, condição limitante para que a área fixa (controle do sistema) seja definida à direita do FPGA e áreas reconfiguráveis à esquerda. Primeiro, posiciona-se a área fixa na região interna mais à direita do FPGA, reservando a região mais à esquerda para as áreas reconfiguráveis. A interface de comunicação, considerada como parte da área fixa, fica posicionada entre cada par de áreas adjacentes. Todos os exemplos apresentados aqui empregam esta organização de planta baixa, mudando eventualmente o número de áreas reconfiguráveis. A Figura 4.8 apresenta um exemplo de leiaute obtido a partir de uma tal definição de áreas para um SDR implementado sobre o FPGA XC2VP30.

Os ambientes EDK e ISE inserem as restrições de definição da planta baixa no arquivo de restrições definido pelo projetista ( o arquivo com extensão UCF, de *User Constraint File*). Deste modo, as restrições definidas para a arquitetura base e armazenadas neste arquivo podem ser reutilizadas no projeto de *arquiteturas secundárias*, assim denominadas por originarem-se da arquitetura base, e diferenciando-se desta apenas por conterem módulos reconfiguráveis distintos daqueles da arquitetura base e funcionarem como geradores de módulos reconfiguráveis, conforme descrito na próxima Seção. Após esta fase, a arquitetura é sintetizada segundo o fluxo descrito na Seção 4.1.7, de modo a gerar o arquivo de configurações do FPGA. Durante o processo de síntese física é gerado também o arquivo contendo informações sobre o posicionamento e o roteamento de recursos do FPGA para a arquitetura projetada. Este arquivo também é reutilizado durante a síntese das arquiteturas secundárias, para que o posicionamento dos recursos de todas as arquiteturas sejam semelhantes.

#### 4.2.2.1 Projeto de Macros para Interface de Comunicação

Com o objetivo de solucionar o problema de dispor de uma infra-estrutura para comunicação entre áreas do SDR Möller propôs macros que possibilitam a definição de uma interface de comunicação em uma posição fixa. As macros são definidas sobre CLBs do FPGA, com a função de transportar sinais de entrada para a saída. As macros projetadas têm capacidade para até 8 sinais. Uma interface de comunicação normalmente possui sinais bidirecionais. Assim, é

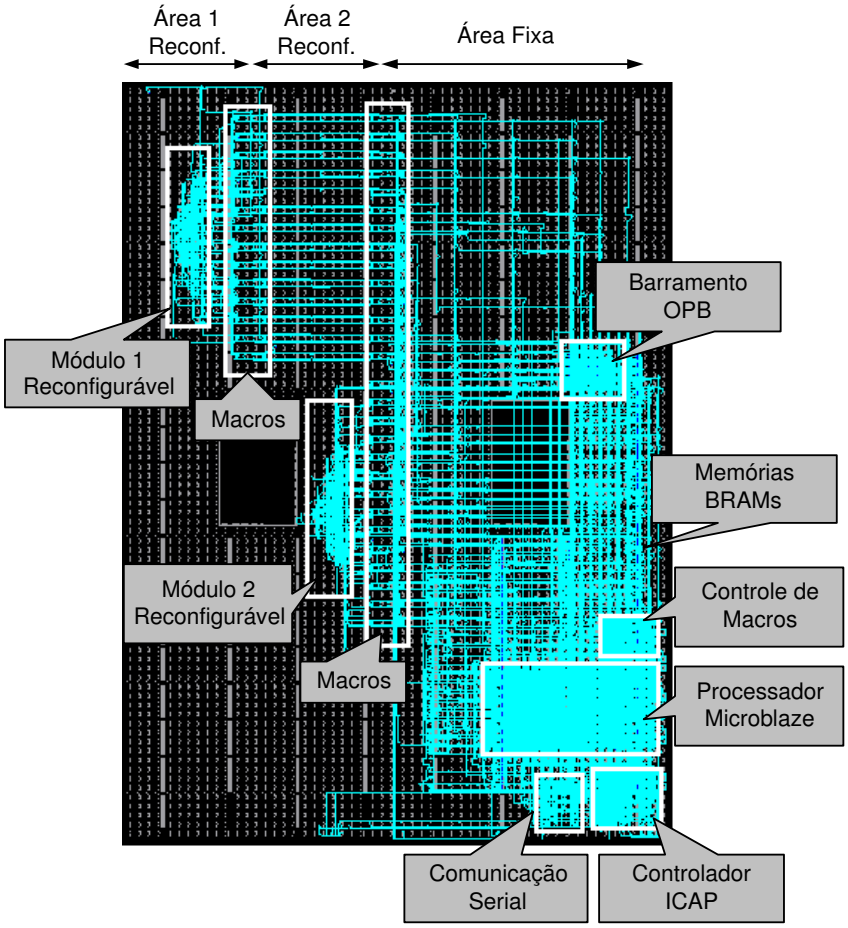


Figura 4.8: Exemplo de leiaute para um SDR implementado em um FPGA XC2VP30 com duas áreas reconfiguráveis, uma área fixa e duas interfaces fixas entre áreas.

necessário construir duas macros diferentes, a Macro RL (do inglês, *Right to Left*) e a Macros LR (do inglês, *Left to Right*), usadas de acordo com o posicionamento relativo das áreas fixa e reconfiguráveis no leiaute do dispositivo. O fato das macros serem definidas sobre CLBs permite que o usuário possa gerar restrições para garantir a posição fixa da interface após qualquer etapa de síntese.

A Macro LR transporta sinais da esquerda para direita de um módulo reconfigurável para o módulo fixo. Esta macro difere da macro RL, pois possui uma chave de controle para desativar os sinais de saída (ver sinal "*chave*"na Figura 4.9). Isto é necessário, para isolar a área de controle do sistema de sinais espúrios gerados pelo módulo reconfigurável durante o processo de reconfiguração, ou seja, assume-se uma posição para a área fixa que é o lado direito do chip. A Figura 4.9 apresenta a implementação das Macros LR e RL por Möller.

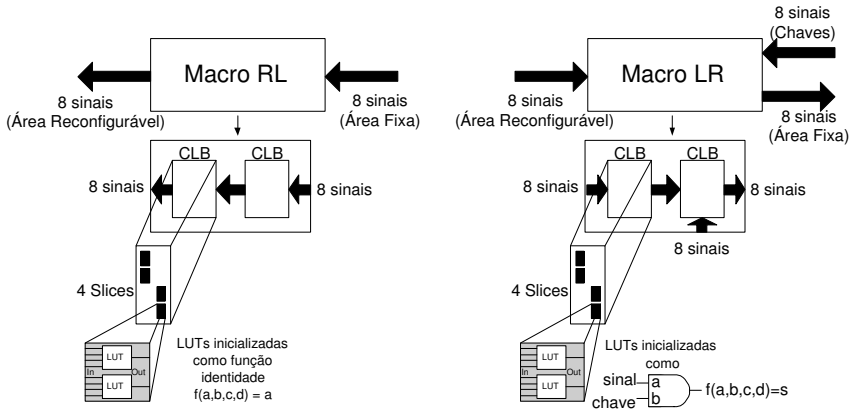


Figura 4.9: Estruturas das Macros RL e LR projetadas por Möller [29].

Como visto na Figura 4.9, a macro LR possui uma porta lógica "And"associando o sinal de entrada com o sinal de controle da chave. Assim, somente existirá variação do sinal na saída da macro caso a chave esteja ativada, ou seja, quando o sinal Chave estiver em nível lógico "1".

A conexão entre módulos reconfiguráveis e o processador MicroBlaze é realizada através do barramento Coreconnect, mais especificamente através do OPB. A interface contém aproximadamente 100 sinais de comunicação. Logo, são necessárias várias macros associadas em paralelo para formar a interface completa.

### 4.2.3 Geração de Bitstreams Parciais

As duas etapas anteriores, Concepção da Arquitetura e Definição de Planta Baixa e Síntese Lógica e Física são repetidas no processo de geração de bitstreams parciais, conforme a Figura 4.6. O processo de definição de novos módulos reconfiguráveis requer a reutilização da arquitetura base concebida inicialmente. Os bitstreams parciais são gerados a partir das arquiteturas secundárias. Cada bitstream parcial deve ser gerado a partir de uma configuração total do FPGA, por um processo de extração. Na configuração de partida, a parte fixa é absolutamente idêntica à parte fixa da arquitetura base e cada uma das áreas reconfiguráveis pode possuir um módulo distinto da arquitetura base. Neste ponto do fluxo de projeto, retorna-se à etapa inicial de concepção e substituem-se os módulos reconfiguráveis da arquitetura base, sem a necessidade de reprojetar integralmente a arquitetura, ou seja, mantendo igual a área fixa do sistema.

Na etapa de definição de planta baixa e síntese lógica e física, reutiliza-se o arquivo UCF com as informações da planta baixa da arquitetura base. Desta maneira, deve-se garantir que as arquiteturas secundárias tenham o mesmo posicionamento relativo de módulos da arquitetura

base. Para que o posicionamento e roteamento de recursos sejam suficientemente semelhantes aos do projeto base, a ferramenta de síntese oferece um mecanismo para sintetizar arquiteturas secundárias exatamente de acordo com o arquivo NCD definido para a arquitetura base. Contudo, o reuso dos arquivos UCF e NCD não é suficiente para evitar roteamentos indesejados de fios na interface das áreas. Portanto, é necessária a verificação visual dos fios próximos à interface, para garantir o isolamento entre as áreas da arquitetura. Com o isolamento garantido, é possível usar a ferramenta CoreUnifier-II Pro para gerar os bitstreams parciais. Esta ferramenta gera bitstreams parciais a partir dos arquivos de configuração obtidos no final do processo de síntese com a ferramenta Bitgen. Esta ferramenta tem como opção a geração de um arquivo padrão ASCII contendo uma cópia do arquivo de configuração binário. A Figura 4.10 representa o processo de aplicação do CoreUnifier-II Pro sobre as arquiteturas para gerar bitstreams parciais. Note-se que a partir da arquitetura base pode-se também gerar bitsreams parciais.

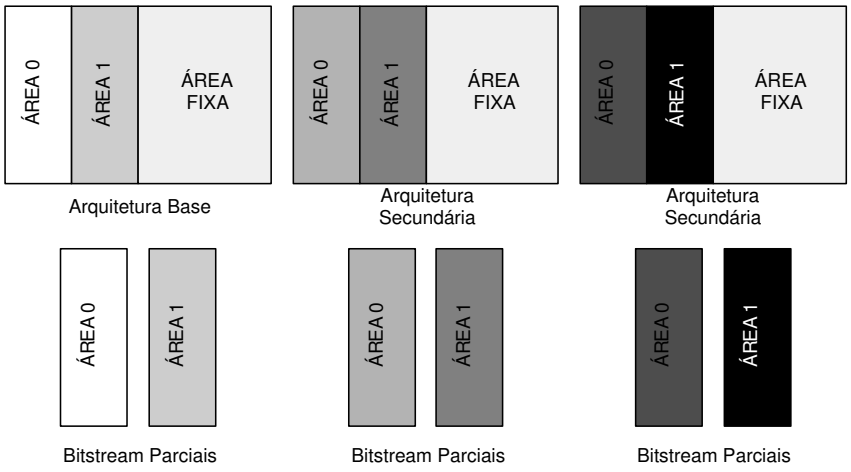


Figura 4.10: Uma representação da geração de bitstreams parciais a partir de arquiteturas desenvolvidas pelos ambientes EDK e ISE.

### 4.2.4 Projeto de Software

Após a conclusão do processo de síntese da arquitetura base no ambiente ISE, o fluxo de projeto de SDRs retorna ao ambiente EDK, para que seja utilizada a plataforma de desenvolvimento disponível. Nesta etapa, é desenvolvido o projeto do software que será executado pelo processador da arquitetura.

## 4.3 Infra-estrutura para Controle do Dispositivo ICAP

Esta Seção descreve a arquitetura de hardware necessária para o controle da porta interna de acesso à configuração (ICAP) disponível em FPGAs da família Virtex. Esta arquitetura oferece suporte à implementação de sistemas auto-reconfiguráveis, alvo de estudo desta dissertação. A arquitetura microprocessada alvo e seus recursos de hardware é apresentada na Figura 4.11.

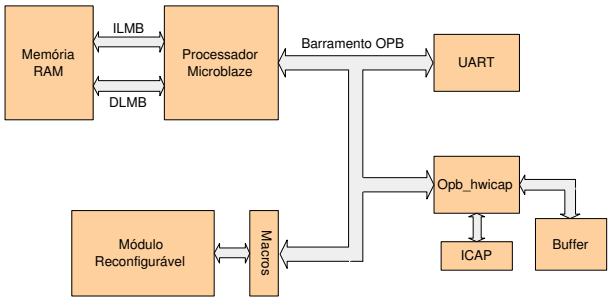


Figura 4.11: Diagrama em blocos da arquitetura alvo para controle do ICAP.

Conforme o fluxo de projeto de SDRs descrito anteriormente, a primeira etapa de projeto é a definição da arquitetura através do fluxo de criação do EDK. Neste caso, como apresentado na Figura 4.11, são utilizados: o processador Microblaze, o barramento Coreconnect com interface OPB, controlador UART (do inglês, *Universal Asynchronous Receiver-Transmitter*) para entrada e saída de dados, memória RAM, controlador *opb_hwicap* para controle do dispositivo ICAP, o módulo reconfigurável e as macros para interface de comunicação. Na etapa seguinte, é construída a planta baixa da arquitetura no ambiente ISE. O resultado desta etapa pode ser visualizado na Figura 4.12. Nesta Figura 4.12, nota-se as macros definindo a interface de comunicação entre as áreas fixa e reconfigurável da arquitetura.

O controlador *opb_hwicap* é o módulo que tem a função de gerenciar o processo de leitura e escrita de palavras na memória de configuração do FPGA. Este controlador é composto basicamente por uma máquina de estados, com a função de gerenciar o processo de leitura e escrita das palavras de configuração, e um buffer de memória, ambos necessários para o acesso à porta ICAP.

Durante o processo de reconfiguração (escrita na memória de configuração do FPGA), a aplicação executada pelo processador busca os bitstreams parciais da memória RAM e os envia ao buffer do controlador *opb_hwicap* através de seu driver. Este buffer tem capacidade para armazenar apenas 512 palavras de 32 bits. Os bitstreams parciais possuem tamanho freqüentemente superior à dimensão do buffer. Portanto, durante o processo de reconfiguração o fluxo de

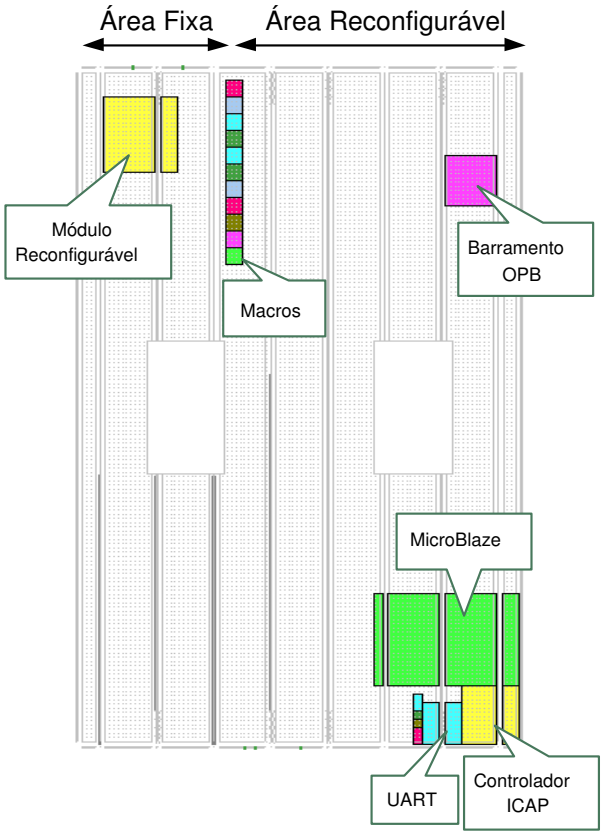


Figura 4.12: Desenvolvimento da planta baixa da arquitetura alvo projetada e prototipada no dispositivo FPGA XC2VP30 da Xilinx.

envio de palavras à ICAP é intercalado pelo processo de preenchimento do buffer do controlador, fazendo com que o processo de reconfiguração sofra interrupções durante a escrita na memória de configurações do FPGA.

### 4.3.1 Driver em Software para Controle do Dispositivo ICAP

Os sistemas auto-reconfiguráveis possuem um processo presente no SDR, que se responsabiliza por alterar o hardware quando isto se faz necessário. Aqui assume-se este processo, denominado controle de configurações, implementado em software, oferecendo o serviço de reconfigurar um módulo de hardware da arquitetura.

O controlador opb_hwicap dispõe de um driver com as funções necessárias aos processos de leitura e escrita de bitstreams através do dispositivo ICAP. No repositório de bibliotecas de drivers do EDK existe o arquivo *opb_hwicap.h* que contém as funções para uso do ICAP [55].

Durante o projeto de software, são utilizadas basicamente 5 funções da biblioteca deste contro-

lador, disponibilizadas na API do sistema: *(i)XHwIcap_ Initialize()*: inicializa uma instância do dispositivo com as características do dispositivo FPGA; *(ii)XHwIcap_ StorageBufferWrite()*: escreve palavra no buffer do controlador; *(iii)XHwIcap_ StorageBufferRead()*: lê palavra do buffer do controlador; *(iv)XHwIcap_ DeviceRead()*: lê palavra do dispositivo ICAP e armazena no buffer; *(v)XHwIcap_ DeviceWrite()*: envia palavra armazenada no buffer ao dispositivo ICAP.

Com a definição deste fluxo de projeto para SDRs e com o apoio dos ambientes de desenvolvimento da Xilinx foi possível projetar uma infra-estrutura para SDRs com capacidade de controle de configurações implementado em software. A estrutura do controlador implementado é tema do próximo Capítulo.

### 4.3.2 Exemplo de Utilização do Driver em Software para Controle do Dispositivo ICAP

Nesta Seção é apresentado um exemplo de emprego das funções presentes no driver do controlador da ICAP e descritas na Seção anterior. Este exemplo pressupõe um bitstream parcial armazenado na memória RAM da arquitetura, o qual será enviado à ICAP através do driver. O código descrito em linguagem C é apresentado a seguir.

```
//bibliotecas que contém funções do driver
#include <xhwicap.h>
#include <xhwicap_i.h>

#define XHI_TARGET_DEVICE           XHI_READ_DEVICEID_FROM_ICAP

//vetor que contém o bitstream parcial descrito
através de palavras de 32 bits representadas no
sistema hexadecimal

int bit_parcial[3833] = {
0xFFFFFFFF,
0xAA995566,
0x30008001,
0x00000007,
0x3001C001,
```

```
...
0x0000000D};

XHwIcap MyIcap; //define uma instância da ICAP

main()
{

...
// Inicializa estrutura MyIcap com informações sobre o FPGA
XHwIcap_Initialize(&MyIcap, XPAR_OPB_HWICAP_0_DEVICE_ID,XHI_TARGET_DEVICE);

...
//armazena no buffer
for(pos=0; pos<512; pos++)
   XHwIcap_StorageBufferWrite(&MyIcap, pos, bit_parcial[pos]);
//envia para ICAP 512 palavras
XHwIcap_DeviceWrite(&MyIcap, 0, 512);

...
//repetir este processo de escrita no buffer e envio de palavras
  à ICAP para todas as palavras do bitstream parcial
...

//armazena no buffer
for(pos=0; pos<512; pos++)
   XHwIcap_StorageBufferWrite(&MyIcap, pos, bit_parcial[pos]);
//envia para ICAP 512 palavras
XHwIcap_DeviceWrite(&MyIcap, 0, 512);
...

}
```

# Capítulo 5

# Proposta de Controlador de Configurações

Os desenvolvimentos na área de sistemas reconfiguráveis têm conduzido a hardware que se assemelha a software do ponto de vista da flexibilidade. Em virtude disso, hardware reconfigurável necessita de subsistemas que executem operações semelhantes às de um sistema operacional. Como descrito no Capítulo 1, um sistema operacional gerencia os processos em uma UCP, controlando a carga de programas em memória a serem executados pelo processador. Para isto, deve interpretar informações provenientes de um escalonador de processos. Em SDRs, deve existir um subsistema que realize tarefas semelhantes a estas.

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

Um dos objetivos específicos deste trabalho é desenvolver um controlador de configurações em software. Esta proposta será baseada na proposta de modelo de controlador de Carvalho [7], sendo de acordo denominada *Reconfigurable System Configuration Manager in Software* (RSCM-S). Este controlador de configurações tem como objetivo suprir carências de infra-estruturas de SDRs como discutido no Capítulo 3, oferecendo os serviços necessários ao gerenciamento de configurações para aplicações que executam sobre o hardware.

A construção de uma nova proposta de controlador de configurações torna possível a evolução e comparação entre as duas implementações do RSCM. Desta forma, é possível medir e anal-

isar o desempenho relativo de ambos controladores enquanto fornecendo serviços às aplicações. As implementações de ambas propostas foram realizadas em dispositivo da família VirtexII-Pro da Xilinx, que permite reconfiguração dinâmica e parcial [61] e encontra-se disponível nos laboratórios do grupo de pesquisa (GAPH da FACIN-PUCRS).

## 5.1 Estrutura Geral do Controlador RSCM-S

A estrutura do controlador de configurações RSCM-S é apresentada no diagrama em blocos na Figura 5.1.

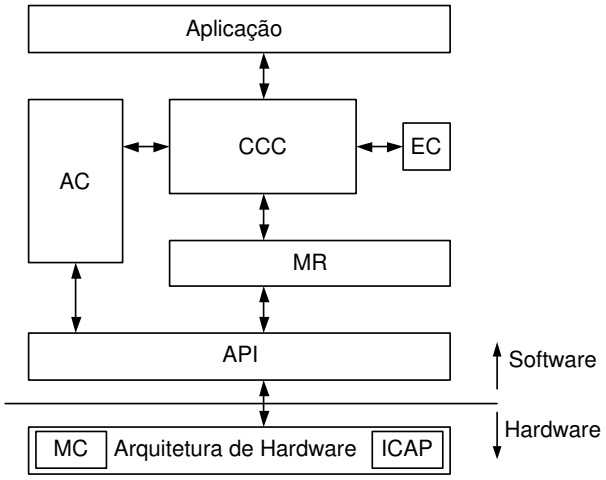


Figura 5.1: Estrutura geral da proposta de controlador de configurações RSCM-S em software baseada na proposta original de Carvalho [7]. A estrutura é composta pelos seguintes módulos: Autoconfigurador (AC), Interface de Programação da Aplicação (API), Controle Central de Configurações (CCC), Escalonador de Configurações (EC), Memória de Configurações (MC), Monitor de Reconfiguração (MR) e a Porta Interna de Acesso à Configuração (ICAP). O controlador de configurações gerencia a arquitetura de um SDR dando suporte a uma dada aplicação.

O controlador RSCM-S possui uma Memória de Configurações (MC), onde serão armazenadas as configurações inativas do sistema. O Autoconfigurador (AC) é responsável por gerenciar o processo de reconfiguração do sistema. O Escalonador de Configurações (EC) controla a ordenação das configurações que deverão ser executadas. O Monitor de Reconfiguração (MR) é responsável por detectar sinais (que indicam o término de execução) dos módulos ativos do sistema. O Controle Central de Configurações (CCC) tem a tarefa de gerenciar e seqüencializar a execução de todos os módulos que formam o controlador proposto. A API disponibiliza uma interface

de comunicação entre o software e o hardware do sistema. O bloco denominado "*Aplicação*"da Figura 5.1 representa o cliente do controlador RSCM-S. A seguir, descreve-se o comportamento e quando for o caso, a estrutura de cada um destes módulos.

### 5.1.1 Monitor de Reconfiguração (MR)

O monitor de reconfiguração (MR) é responsável por detectar situações que disparam o processo de reconfiguração em uma aplicação. Nesta proposta, existem dois diferentes casos em que o monitor é empregado. No primeiro caso, as configurações, ao concluírem seu processamento geram sinais que são detectados pelo monitor. O monitor identifica a configuração que sinaliza o evento e repassa esta informação ao CCC. No segundo caso, o monitor é colocado em um ponto estratégico do software da aplicação. Isto faz com que a aplicação possa decidir o momento em que o processamento de uma configuração é concluído.

Esta proposta de controlador de configurações não prevê preempção nem salvamento de contexto de configurações, caso em que uma configuração tem seu processamento interrompido temporariamente, cedendo sua área para que outra de maior prioridade seja executada, e possa retornar ao sistema e continuar seu processamento.

Os sinais gerados no primeiro caso e a execução de uma instrução no segundo caso, determinam a conclusão de tarefas realizadas por uma configuração, conseqüentemente liberando a área reconfigurável para que outra configuração seja empregada na mesma área.

### 5.1.2 Escalonador de Configurações (EC)

O escalonador de configurações (EC) tem a função de armazenar um cronograma estático de execução das configurações. Este cronograma é definido segundo uma política de escalonamento fixa, ou seja, definida em tempo de projeto do sistema. Nesta versão do RSCM, ainda não há suporte a escalonamento dinâmico.

Este módulo possui uma estrutura de dados onde estão definidas as dependências entre as configurações. Esta estrutura é chamada de *Tabela de Dependência e Descritores* (TDD) conforme definido em [7]. Uma TDD exemplo é mostrada na Tabela 5.1, seu conteúdo representa o grafo de configurações da Figura 5.2.

Este exemplo apresenta a definição de precedências de execução de configurações para uma dada aplicação. Através do grafo nota-se que uma configuração ao concluir sua tarefa pemite que novas configurações sejam utilizadas pelo sistema. Neste exemplo a TDD requer 3 campos para

Tabela 5.1: Estrutura da Tabela de Dependência e Descritores (TDD) do Escalonador de configurações, baseada em [7]. O campo *config* indica o identificador da configuração; *Npredec* indica o número de predecessores da configuração; Suc $N$ representa o sucessor $N$ da configuração.

| Config. | Npredec. | Suc. 1 | Suc. 2 | Suc. 3 |
|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 3 |
| 1 | 1 | 4 | 5 | - |
| 2 | 1 | 6 | - | - |
| 3 | 1 | 6 | 7 | 8 |
| 4 | 1 | - | - | - |
| 5 | 1 | - | - | - |
| 6 | 2 | 9 | - | - |
| 7 | 1 | - | - | - |
| 8 | 1 | 0 | - | - |
| 9 | 1 | - | - | - |

identificar as sucessoras de cada configuração. O número de sucessoras independe do número de áreas reconfiguráveis. Para cada aplicação o número de sucessoras é definido em tempo de projeto, consequentemente originando uma nova TDD.

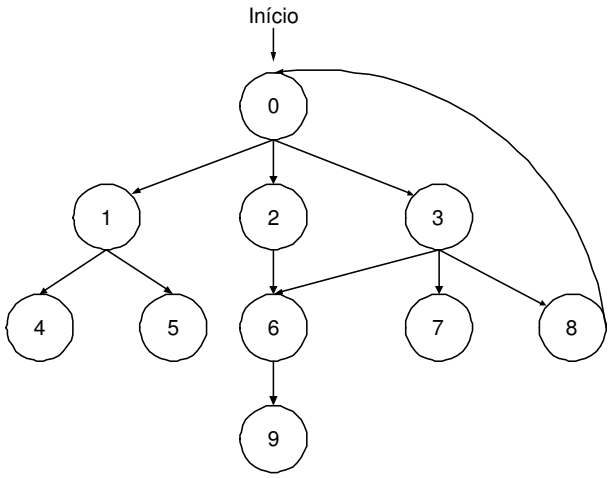


Figura 5.2: Grafo de dependências de configurações utilizado para criar a TDD da Tabela 5.1.

As linhas da tabela correspondem ao número de configurações parciais do sistema. Para cada projeto de sistema deve haver uma nova TDD, que determine as dependências entre as configurações. As dimensões da tabela são determinadas pelo número de configurações e das dependências existentes entre elas.

Em arquiteturas definidas com apenas uma área reconfigurável o EC utiliza uma estrutura de fila para o armazenamento das configurações. Deste modo, durante o projeto do sistema, o projetista define a ordem de execução das configurações preenchendo a fila com a identificação das configurações para que o escalonador utilize durante a execução.

### 5.1.3 Memória de Configuração (MC)

MC tem como função armazenar todas as configurações do sistema durante sua execução. Esta memória pode ser interna ou externa ao FPGA, dependendo apenas do número e tamanho das configurações parciais do sistema. Os bitstreams são compactados antes de serem armazenados, visando otimizar a ocupação da memória.

### 5.1.4 Autoconfigurador (AC)

O autoconfigurador (AC) é o módulo responsável por gerenciar o processo de reconfiguração propriamente dito. Este módulo recebe do CCC informações sobre a configuração e área onde esta deverá ser configurada. O AC busca o bitstream da memória de configurações e o envia para a porta de configuração do dispositivo (ICAP). Os bitstreams compactados em memória devem ser descompactados pelo AC. O processo de envio dos bitstreams ao ICAP utiliza as funções disponíveis na API para acessar o controlador da ICAP em hardware.

### 5.1.5 Controle Central de Configurações (CCC)

O Controle central de configurações (CCC) tem a função de gerenciar os diversos módulos do controlador RSCM-S, aplicando o cronograma que recebe do módulo escalonamento de configurações. A implementação deste controlador será em software sob uma arquitetura monoprocessada, portanto a execução dos módulos será seqüencial. Esta seqüência é dada pelo recebimento de requisições do monitor de reconfiguração, seguido pela requisição de serviços do escalonador de configurações e do autoconfigurador, de modo a obter-se uma nova configuração na arquitetura.

### 5.1.6 Interface de Programação da Aplicação (API)

A proposta de implementação do controlador de configurações em software requer uma interface hardware/software que ofereça suporte ao controle do hardware.

Este módulo emprega várias funções disponíveis no repositório de *drivers* do ambiente EDK [55].

### 5.1.7 Algoritmos para Compactação e Descompactação de Bitstreams

Os arquivos de configuração parciais gerados pela ferramenta Bitgen são conjunto de palavras de 32 bits representadas em binário no formato .rbt, estas mesmas palavras aparecem como cadeias de caracteres ASCII representadas em notação hexadecimal.

Visando reduzir a quantidade de memória necessária para armazenar bitstreams no sistema embarcado, algum esforço foi despendido na construção de uma ferramenta de compactação. Uma observação de numerosos bitstreams de FPGAs de várias famílias da Xilinx revelaram uma alta freqüência dos padrões hexadecimais de 32 bits *0x00000000* e *0x20000000*. A partir desta, implementou-se uma ferramenta de compactação que detecta estes padrões e conta o número de repetições encontradas em seqüências dentro do arquivo. Após a contagem, o algoritmo substitui a seqüência por uma palavra-chave que contenha o número de repetições do padrão. A palavra-chave deve ser um número hexadecimal que tenha um valor diferente dos encontrados em bitstreams. Optou-se por usar as seguintes palavras-chaves:

- 1234xxxx para o padrão 0x00000000

- 4321xxxx para o padrão 0x20000000

Os caracteres *xxxx* representam o número de repetições do padrão. O número de repetições é representado também no sistema hexadecimal.

Obviamente, é possível existirem no bitstream original palavras que iniciam com um destes prefixos (1234/4321). Portanto, quando identificados são inseridos no bitstream compactado a palavra de controle *0x12340000* de maneira a anteceder o padrão originalmente encontrado. Desta forma, viabiliza-se a utilização dos prefixos escolhidos.

O algoritmo de compactação utilizado para construir a ferramenta é apresentado na Figura 5.3.

Uma ferramenta para descompactar bitstreams foi construída com o intuito de regenerar os bitstreams originais a partir da versão compactada. O algoritmo desenvolvido para a descompactação reconhece as palavras-chaves e as substitui por uma série contínua do padrão identificado. Desta maneira, pretende-se obter novamente o bitstream em seu formato original para ser usado no processo de reconfiguração do dispositivo FPGA.

O descompactador, ao contrário do Compactador, é parte do sistema embarcado. A ferramenta de descompactação é usada na implementação do módulo AC do controlador RSCM-S

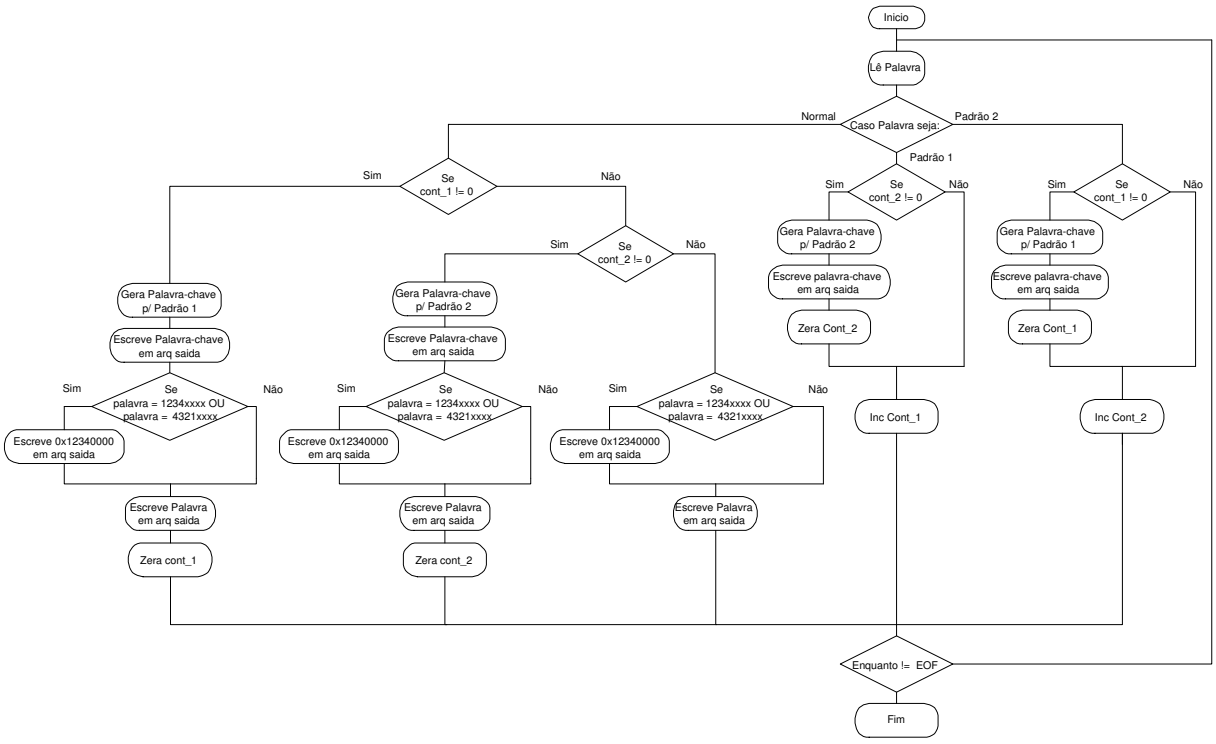


Figura 5.3: Fluxograma do algoritmo da ferramenta de compactação de bitstreams.

descrito, na Seção 5.2. Portanto, o algoritmo deve prever uma restrição do hardware do sistema: o tamanho do buffer do controlador opb_hwicap para o envio de palavras ao ICAP.

Com base nesta restrição construiu-se um algoritmo que descompacta o bitstream em blocos de tamanho máximo igual a 512 palavras. O algoritmo deve garantir que nenhuma palavra seja perdida durante a descompactação, para que todas sejam enviadas ao ICAP. O fluxograma do algoritmo é apresentado na Figura 5.4.

#### 5.1.7.1 Resultados Obtidos com o Algoritmo

Após a implementação deste algoritmo trivial de compactação, obteve-se alguns resultados, apresentados na Tabela 5.2.

Observando os resultados conclui-se que a ferramenta atinge taxas de compactação em torno de 90% para casos onde a ocupação do FPGA seja pequena ($< 10$ %), conforme os exemplos apresentados na Tabela 5.2. Este algoritmo naturalmente não é eficiente para bitstreams de projetos que ocupam grande parte do FPGA, fato que reduz a ocorrência do padrão *"0x00000000"* no bitstream.

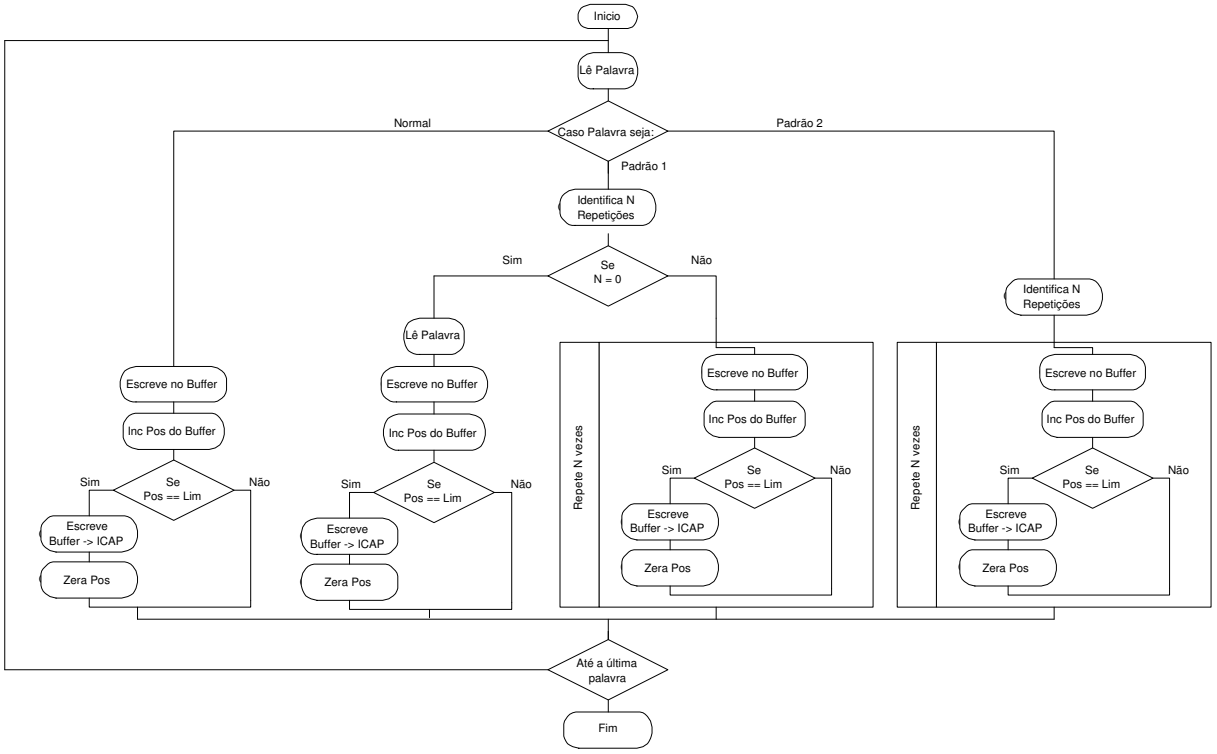


Figura 5.4: Fluxograma do algoritmo de descompactação de bitstreams utilizado no módulo Autoconfigurador do RSCM-S.

## 5.2 Implementação do RSCM-S

### 5.2.1 Memória de Configuração (MC)

A memória de configurações deve ter tamanho suficiente para armazenar todos as configurações inativas de um dado SDR. As implementações foram realizadas no dispositivo XC2VP30 da família VirtexII-Pro da Xilinx embarcado na plataforma VirtexII-Pro$^{TM}$FF1152 da Memec Insight [26]. Esta plataforma oferece duas possibilidades de armazenamento: BRAMs internas ao dispositivo FPGA e SDRAMs disponíveis na plataforma. A Tabela 5.3 apresenta os tipos e quantidades de memória disponíveis.

Nos experimentos realizados não houve a necessidade de armazenar dados em SDRAM, pois com a compactação e o reduzido número de bitstreams parciais usados nos projetos foi suficiente utilizar apenas BRAMs. O armazenamento em SDRAMs requer um subsistema em software de carga de configurações para inicializar as SDRAMs com bitstreams parciais. O uso deste subsistema é necessário devido à restrição da ferramenta de configuração, que permite apenas a inicialização das memórias BRAMs. Como a capacidade de armazenamento das BRAMs foi

Tabela 5.2: Resultados obtidos com a aplicação da ferramenta de compactação em arquivos de configurações. A unidade KB representa K bytes. Os arquivos denominados "Projetos"representam uma configuração total de um FPGA, sendo *Contador* relativo ao FPGA XC2VP30 e *Multinoc* relativo ao FPGA Spartan2. Os arquivos restantes representam configurações parciais relativas ao FPGA XC2VP30. A taxa de ocupação é medida em relação a área reservada à lógica implementada.

| Arquivo | Número de palavras | Tamanho | Número de Palavras após compactação | Tamanho Compactado | Ocupação. | Taxa de Compactação |
|---|---|---|---|---|---|---|
| Porta Or | 636 | 2.5 KB | 34 | 136 B | - | 94,65% |
| Contador Pequeno | 72737 | 284 KB | 3754 | 14 KB | 3,8% | 94,8% |
| Projeto Contador | 362185 | 1.4 MB | 27265 | 107 KB | 8% | 92,47% |
| Projeto Multinoc | 45063 | 176 KB | 39890 | 156 KB | 97% | 11,48% |

Tabela 5.3: Tamanho das memórias disponíveis na plataforma VirtexII-Pro$^{TM}$FF1152 [26].

| Tipo de Memória | Tamanho |
|---|---|
| BRAM | 306 KB |
| SDRAM | 64MB |

suficiente, optou-se por não implementar tal subsistema de inicialização e transferência de bitstreams.

### 5.2.2 Interface de Programação da Aplicação (API)

A estrutura do RSCM-S é formada por um módulo responsável por estabelecer a comunicação entre a infra-estrutura de hardware e o restante do controlador de configurações. Este módulo é denominado API, tendo a função de oferecer mecanismos para que através de operações realizadas em software o RSCM-S gerencie a infra-estrutura de hardware.

A implementação da API reúne diversas funções de biblioteca do EDK para controle do dispositivo ICAP e acesso a periféricos. Na implementação do controlador RSCM-S foram empregadas as seguintes funções apresentadas na Tabela 5.4, que compõem a API do RSCM-S.

### 5.2.3 Monitor de Reconfiguração (MR)

Na proposta do controlador RSCM-S, o monitor de reconfiguração identifica situações em um SDR que identificam a ocorrência de reconfiguração, disparada pelo hardware e disparada pelo software. O monitor pode atuar de duas formas possíveis, como descrito anteriormente.

Tabela 5.4: Funções disponíveis na API do RSCM-S.

| | |
|---|---|
| XHwIcap_StorageBufferWrite() | escreve palavra no buffer. |
| XHwIcap_StorageBufferRead() | lê palavra do buffer do controlador. |
| XHwIcap_DeviceRead() | lê palavra do ICAP e armazena no buffer. |
| XHwIcap_DeviceWrite() | envia palavra do buffer ao dispositivo ICAP. |
| XMycore_mWriteReg() | escreve palavra no periférico. |
| XMycore_mReadReg() | lê palavra do periférico. |

No primeiro caso, o monitor verifica sinais que indicam o término de processamento proveniente dos módulos reconfiguráveis. No segundo do caso, o monitor é posicionado estrategicamente no código da aplicação, gerando em software o disparo do processo de reconfiguração.

No primeiro caso, a infra-estrutura de hardware do sistema deve permitir a comunicação entre módulos reconfiguráveis e processador através de interrupções. Com este suporte de hardware, os módulos reconfiguráveis ao concluírem suas tarefas geram uma requisição de interrupção ao processador do sistema. Este deverá desviar seu processamento para uma rotina de tratamento de interrupção, onde se encontra a implementação em software do controlar RSCM-S. Neste caso, o monitor utiliza uma função da API para ler o código de identificação do módulo solicitador da interrupção. Esta identificação é repassada ao controle central de configurações, seguindo o processamento no RSCM-S.

O código de identificação de cada módulo reconfigurável encontra-se em um registrador. Este registrador é lido através da função da API *XMycore_ mReadReg()*. Esta função contém parâmetros que identificam o módulo e o registrador do módulo que possui o código de identificação. Uma chamada desta função tem a aparência abaixo.

```
XMycore_mReadReg(ENDBASE, ENDREG);
```

Nesta linha, ENDBASE representa o endereço do módulo reconfigurável e ENDREG o endereço do registrador.

No segundo caso, o monitor é representado estaticamente no código do software da aplicação por uma chamada de procedimento. O controlador RSCM-S, neste caso, encontra-se em um procedimento da aplicação. Este procedimento é invocado através do monitor, que passa como parâmetro o código do módulo que deve ser retirado do sistema. Neste caso, em tempo de projeto são previstos todos os pontos onde deverão ser realizadas as reconfigurações do sistema, colocando-se os monitores com os respectivos módulos que devem ser substituídos. Um exemplo

de uso do monitor para este caso é dado abaixo:

*rscm-s(2);*

Neste código, *rscm-s()* representa o nome do procedimento e *2* representa o número da configuração que deve ser substituída pelo RSCM-S.

O monitor que representa o segundo caso foi implementado em infra-estruturas com uma e duas áreas reconfiguráveis. Já o monitor que representa o primeiro caso, foi implementado apenas em infra-estruturas com uma fatia reconfigurável, pois o hardware de suporte para interrupção torna ainda mais crítica a definição de áreas reconfiguráveis no hardware. Mais detalhes destas implementações estão contidas no Capítulo 6 deste documento.

### 5.2.4 Autoconfigurador (AC)

Este módulo do RSCM-S tem grande importância no desempenho do sistema. O autoconfigurador tem a função de controlar o processo de reconfiguração parcial propriamente dito do dispositivo. O RSCM-S evoluiu a proposta de Carvalho [7], permitindo que o próprio sistema reconfigure-se através da ICAP. Esta característica aumenta a autonomia do SDR.

No RSCM-S, o autoconfigurador recebe do controle central de configurações a identificação da configuração e a fatia onde deve ser configurada no sistema. No caso de arquiteturas com apenas uma área reconfigurável, obviamente não é necessário identificar a fatia que se deseja configurar. O autoconfigurador busca em memória a configuração e a envia ao buffer do controlador da ICAP. Um vetor indexado pela identificação de cada configuração realiza o mapeamento do endereço de memória das configurações. A estrutura do vetor é mostrada a seguir.

*mapa_de_memoria[id_code].inicio*

*mapa_de_memoria[id_code].tam*

Neste código, *id_code* representa o código de identificação da configuração, *inicio* representa a posição de memória inicial da configuração e *tam* representa o número de posições ocupadas na memória a partir de *inicio*.

Como as configurações encontram-se compactadas na memória, deve ocorrer o processo de descompactação durante o envio ao buffer do controlador da ICAP, conforme a Seção 5.1.7. O autoconfigurador transfere os blocos armazenados no buffer do controlador à ICAP, repetindo

este processo até a transmissão completa da configuração, realizando desta forma a reconfiguração do dispositivo.

### 5.2.5 Escalonador de Configurações (EC)

Este módulo detém a política utilizada para escalonar as configurações do sistema. O controlador RSCM-S adota uma política de escalonamento estática, ou seja, um cronograma é definido em tempo de projeto em forma de uma TDD para as configurações, na qual o escalonador se baseia durante a execução do sistema [7].

Para implementar em software o escalonador, foi criada uma estrutura de dados que representa a TDD.

A implementação da TDD foi realizada com a seguinte estrutura de dados:

```
struct tabela{  int n_predec;
                int suc1;
                int suc2;} tdd[CONFIG];
```

Nesta estrutura, *CONFIG* representa o código de identificação de uma configuração, *n_predec* representa o número de predecessoras, *suc1* e *suc2* armazenam a identificação das configurações sucessoras.

A implementação da fila de execução usa o código abaixo.

*fila_escalonadas[n_configs];*

Neste, *n_configs* representa o número total de configurações do sistema.

### 5.2.6 Controle Central de Configurações (CCC)

O controle central de configurações é responsável por sincronizar e estabelecer o ordenamento de funcionamento dos módulos que compõem o RSCM-S. O CCC contém uma tabela de alocação de recursos (TAR), onde mantém informações sobre a localização das configurações e sobre as áreas reconfiguráveis da arquitetura. Uma fila de configurações já executadas também é utilizada, para que o CCC tenha controle sobre a execução das configurações.

A fila de controle das configurações executadas foi implementada sobre um fila com tamanho igual ao número de configurações utilizadas pelo sistema. A estrutura é mostrada a seguir.

*fila_executadas[n_configs]*

A tabela de alocação de recursos (TAR) é utilizada apenas em arquiteturas com duas ou mais áreas reconfiguráveis. Em arquiteturas com uma área reconfigurável apenas uma variável controla a configuração alocada na fatia reconfigurável. A estrutura que implementa a TAR é mostrada a seguir.

*tar[n_fatias]*

Nesta, *n_fatias* representa o número de áreas reconfiguráveis do sistema.

O fluxo seqüencial de funcionamento da atual implementação é realizado em 6 etapas descritas abaixo.

1. o monitor de reconfiguração envia a identificação (ID) da configuração parcial do bitstream parcial ao CCC;
2. o CCC coloca a ID na fila de configurações executadas;
3. o CCC requisita o serviço do escalonador, enviando a ID ao mesmo;
4. o escalonador devolve ao CCC a ID da próxima configuração a ser configurada;
5. o CCC solicita o serviço do autoconfigurador (AC), enviando a ID da próxima configuração;
6. o CCC atualiza a tabela TAR com o novo recurso.

# Capítulo 6

# Estudos de Caso

Este Capítulo descreve os estudos de caso realizados para validação da infra-estrutura de hardware para controle de configurações em software proposta e implementada. Ele também descreve a validação da proposta do controlador RSCM-S. A partir destes estudos de caso foram realizadas análises preliminares para avaliação de desempenho do sistema, visando a comparação com a proposta de controle de configurações de Carvalho [7].

Neste Capítulo são apresentados cinco estudos de caso usados para validar a infra-estrutura de hardware para SDRs em conjunto com o controlador RSCM-S. A Seção 6.1 apresenta quatro estudos de caso simples, que demonstram a efetividade do fluxo de projeto proposto nas Seções 4.2 a 4.3.1 para o desenvolvimento de infra-estruturas de SDRs. A Seção 6.2 descreve o estudo de caso de uma aplicação exemplo para a infra-estrutura projetada. Finalmente, na Seção 6.3 são realizadas medições e comparações preliminares das propostas de controle de configuração RSCM e RSCM-S.

## 6.1 Validação da Proposta de Infra-estrutura para SDRs

Nesta Seção são apresentados os projetos empregados para prova de conceito e validação da proposta de infra-estrutura para controle de configurações em software.

### 6.1.1 Estudo de Caso 1: Projeto And_Or

O projeto And_Or foi usado para realizar a prova de conceito da estrutura de controle da porta interna de configuração (ICAP). Este projeto utiliza uma área reconfigurável onde se implementa um módulo contendo apenas uma porta lógica *And*. Através do fluxo das diferenças

da Xilinx [60] é gerado um bitstream parcial para reconfigurar a área para executar a função de uma porta lógica *Or*.

#### 6.1.1.1 Estrutura do Sistema

O projeto And_Or foi integralmente elaborado utilizando as ferramentas dos ambientes EDK e ISE. A estrutura geral do sistema é apresentada na Figura 6.1. Neste, emprega-se o processador Microblaze, juntamente com recursos para comunicação serial entre o FPGA e um PC, recursos para controle da porta ICAP, além do módulo reconfigurável e memória para processamento. Para realizar medições de tempo de execução de funções realizadas pelo sistema, foi criado e alocado um módulo *Temporizador* em hardware, com a função de contar ciclos de relógio em instantes pré-determinados. O Temporizador é habilitado/desabilitado por software. O analisador lógico disponível em laboratório não possui a capacidade de memória para realizar medições de tempo longas (relógio na ordem de ns e intervalos da ordem de ms). Assim, foi necessário implementar em hardware o Temporizador e adicioná-lo ao sistema.

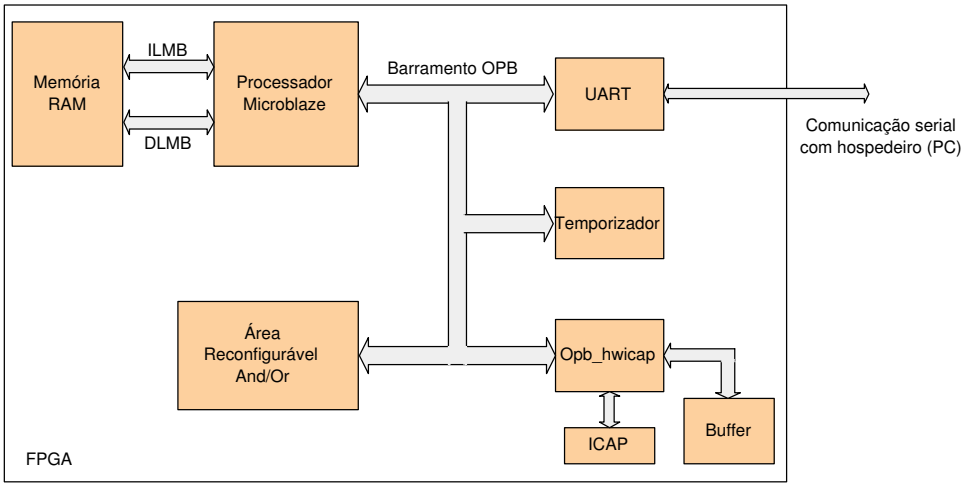


Figura 6.1: Diagrama de blocos do projeto And_Or.

Os módulos Temporizador e a Área Reconfigurável foram construídos a partir do fluxo de criação de periféricos do ambiente EDK, como visto no Capítulo 4. A Área Reconfigurável foi projetada com um registrador de 32 bits, onde se implementa uma porta lógica entre alguns bits deste registrador, conforme ilustrado na Figura 6.2. A porta lógica reconfigurável usa 3 bits do registrador, sendo dois bits as entradas e um bit a saída da porta.

O software executado pelo processador MicroBlaze no sistema And_Or foi implementado para realizar operações de escrita e leitura no registrador do módulo reconfigurável, bem como

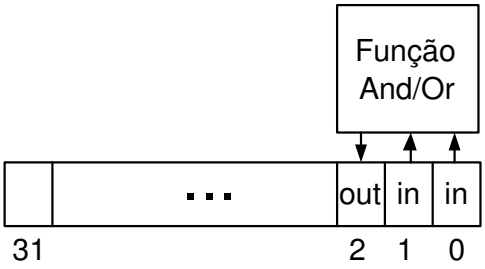


Figura 6.2: Estrutura Geral da Área Reconfigurável para o projeto And_Or.

para controlar o processo de reconfiguração do sistema. Além disso, este software realiza o envio de dados serialmente ao PC para que seja possível visualizar a operação do sistema. O processamento visa comprovar o funcionamento da lógica *and* para todas combinações possíveis de entrada.

O projeto de software possui um procedimento que realiza o processo de reconfiguração parcial do sistema, através da configuração inicialmente armazenada na memória do sistema. Este procedimento é executado em um determinado momento para que a MicroBlaze realize, através da porta ICAP, a reconfiguração parcial transformando a porta lógica *And* em uma *Or* ou vice-versa.

#### 6.1.1.2 Resultados

Concluída a implementação deste projeto, foi possível analisar o sistema segundo a ocupação de área do FPGA e o tempo necessário para realizar o processo de reconfiguração parcial. O projeto And_Or foi prototipado no dispositivo XC2VP30, onde o sistema foi projetado para operar a uma freqüência de relógio de 25MHz.

Com a realização da síntese da arquitetura obteve-se como resultado o posicionamento dos módulos da arquitetura sobre o dispositivo mostrado na Figura 6.3. A partir desta Figura e de relatórios gerados pelas ferramentas do ambiente ISE, foi possível determinar a área ocupada pela arquitetura sobre o FPGA. A Tabela 6.1 apresenta os recursos utilizados para implementar cada um dos módulos que compõem a arquitetura.

A realização do mapeamento da interface ICAP para pinos de I/O do FPGA permitiu a análise destes sinais com um analisador lógico Agilent 1672G. Desta forma, foi possível obter as formas de onda destes sinais durante o envio de dados ao ICAP, bem como, a medição do tempo necessário para o preenchimento do buffer do controlador OPB_hwicap e o tempo necessário

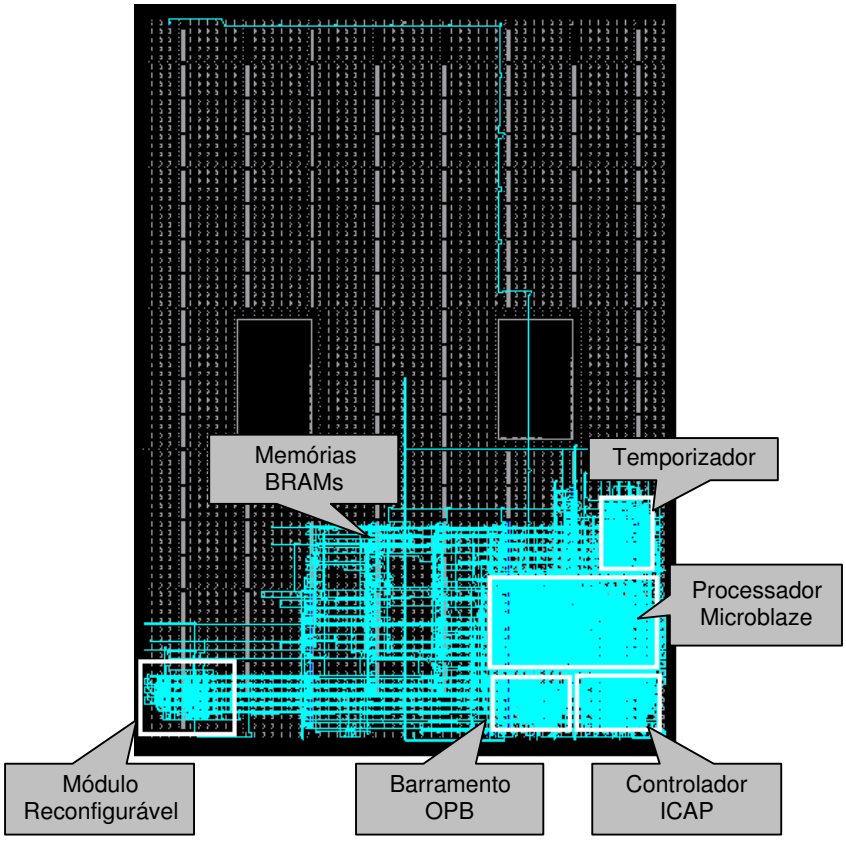


Figura 6.3: Leiaute resultante da do projeto And_Or, conforme visualizado com a ferramenta FPGA Editor.

para o envio de palavras do buffer à ICAP.

Neste estudo de caso, foram realizadas medições de tempo para preenchimento do buffer do controlador ICAP, conforme apresentado na Tabela 6.2. Uma medição realizada com o uso do descompactador de bitstreams e outra sem o descompactador. O tempo de envio de dados do buffer à porta ICAP é independente do uso de compactação de bitstreams.

A Figura 6.4 mostra parte da sinalização relevante do processo de reconfiguração do FPGA através do ICAP, conforme observado através do analisador lógico. Nesta Figura é apresentada a variação de sinais relevantes durante a escrita no ICAP do primeiro lote de 512 palavras do bitstream parcial, bem como o tempo para carregar o buffer do controlador OPB_hwicap com as 124 palavras restantes do bitstream. O tempo de carga do buffer com as 512 palavras iniciais do bitstream não é visualizado neste instante, mas deve ser incluído na latência de chaveamento da configuração.

Foram realizadas medições com e sem uso da rotina de descompactação. Assim, foi pos-

Tabela 6.1: Utilização dos recursos do FPGA para implementar o projeto And_Or sobre o dispositivo XC2VP30.

| Módulo | Número de Slices | Número de DFFs | Percentual de Slices |
|---|---|---|---|
| Microblaze | 571 | 366 | 4,17% |
| MB_OPB | 89 | 11 | 0,65% |
| OPB_hwicap | 151 | 155 | 1,1% |
| DLMB | 2 | 1 | 0,01% |
| DLMB_Cntlr | 3 | 1 | 0,02% |
| ILMB | 2 | 1 | 0,01% |
| ILMB_Cntlr | 1 | 1 | 0,007% |
| RS-232 | 63 | 60 | 0,45% |
| Temporizador | 151 | 196 | 1,1% |
| Módulo Reconfigurável | 90 | 135 | 0,66% |
| Área Total | 1123 | 927 | 8,20% |

Tabela 6.2: Resultados de medições realizadas para avaliar o desempenho do processo de reconfiguração parcial no projeto And_Or.

| **Tipo de medição** | **Tempo** |
|---|---|
| Escrita de 512 palavras no buffer (sem descompactação) | 1,293ms |
| Escrita de 512 palavras no buffer (com descompactação) | 1,378ms |
| Envio de 512 palavras à ICAP | 81,88$\mu$s |
| Tempo de reconfiguração total para 636 palavras sem descompactação | 1,710ms |
| Tempo de reconfiguração total para 636 palavras com descompactação | 1,811ms |

sível perceber que os tempos de carga do buffer sofrem um aumento, devido ao processo de descompactação que atinge algumas dezenas de microssegundos.

### 6.1.2 Estudo de Caso 2: Projeto Contador Up-Down

O projeto Contador Up-Down tem como objetivo validar o fluxo de projeto proposto no Capítulo 4, e validar o funcionamento de SDRs com suporte a comunicação entre processador e periféricos através de interrupção. Este estudo de caso também serve para validar a utilização das macros propostas por Möller [29], que definem a interface de comunicação entre áreas fixa e reconfiguráveis.

A arquitetura de hardware deste projeto é basicamente composta pelos mesmos recursos

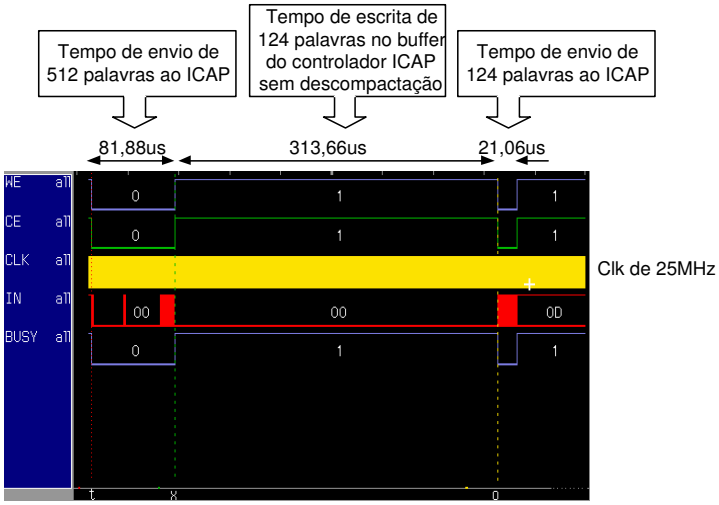


Figura 6.4: Sinalização gerada durante o envio de 512 palavras de 32 bits à ICAP, conforme observado na tela do analisador lógico Agilent 1672G.

do Estudo de Caso 1. Adicionalmente, contém hardware para controle de interrupções e utiliza contadores progressivos e regressivos implementados no módulo reconfigurável, ao invés de apenas portas lógicas. Uma unidade de controle para o acionamento das macros também é utilizada neste projeto. A Figura 6.5 apresenta um diagrama de blocos da arquitetura usada para implementar este estudo de caso.

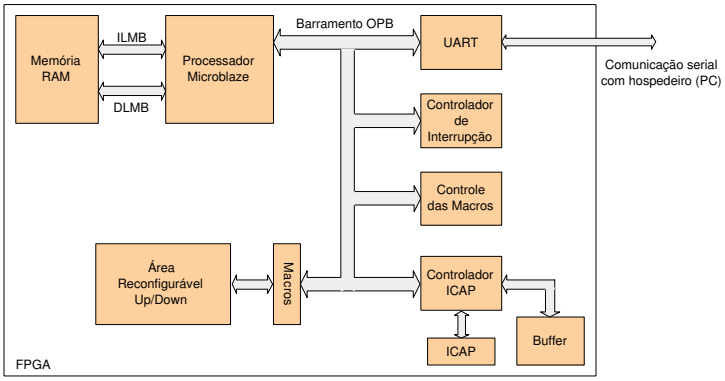


Figura 6.5: Diagrama em blocos do projeto Contador Up-Down.

Inicialmente, o projeto possui o módulo reconfigurável operando como um contador progressivo. Durante a execução o contador, ao atingir o valor "*8*", gera um sinal de interrupção ao processador, fazendo com que este desvie seu fluxo de execução para uma *rotina de tratamento de interrupção*, onde se encontra o código do controlador de configuração RSCM-S.

A requisição de interrupção causada pelo módulo de contagem sinaliza a conclusão de seu

processamento, ou seja, informa ao controlador de configurações (através do monitor de reconfiguração, MR) que a área está livre para receber outra configuração de hardware. Com isto, o controlador determina (via módulos CCC/EC/AC) a próxima configuração e reconfigura o módulo contador, para que este opere como contador regressivo. Após este processamento, o RSCM-S encerra sua operação, permitindo que o processador retorne ao seu fluxo de execução. Durante este processo não exite salvamento do conteúdo do registrador ou uso de reset no módulo contador.

O software projetado para este estudo de caso monitora o processo de contagem através de funções da API e transmite serialmente ao PC os dados de contagem obtidos. Desta forma, é possível verificar e validar o funcionamento do sistema através de um hiperterminal disponível no sistema operacional do hospedeiro.

#### 6.1.2.1 Resultados

Com a implementação do projeto Contador Up-Down foi possível validar o fluxo de projeto, juntamente com a interface de comunicação definida via macros. Com as ferramentas do ambiente ISE, foram obtidos relatórios e a descrição da alocação e roteamento dos módulos que compõem o projeto. A Figura 6.6 apresenta a estrutura física (leiaute) do sistema sobre o dispositivo XC2VP30.

Nesta Figura, é possível visualizar a interface IPIF, composta pelas macros que transportam sinais, isolam as áreas fixa e reconfigurável e estabelecem pontos de passagem para estes no leiaute do FPGA. Como cada macro pode transportar até 8 sinais e a interface IPIF é constituída, neste caso, por 104 sinais, faz-se necessário o uso de 14 macros para estabelecer a interface de comunicação. O módulo Controle de Macros indicado na Figura 6.6 controla o acionamento da chave que habilita/desabilita a passagem de sinais nas macros LRC durante o processo de reconfiguração, evitando a ação de sinais espúrios sobre o processador.

A Tabela 6.3 apresenta os recursos do FPGA utilizados para implementar a arquitetura do projeto Contador Up-Down sobre o dispositivo XC2VP30.

Definindo-se Área de Controle como sendo o conjunto de todos os módulos que compõem a arquitetura com exceção da área reconfigurável, pode-se notar na Tabela 6.3 um aumento desta área em relação ao Estudo de Caso 1, devido ao uso dos módulos de controle das macros e o controle de interrupções.

Este estudo de caso utiliza apenas uma configuração parcial/área reconfigurável (contador

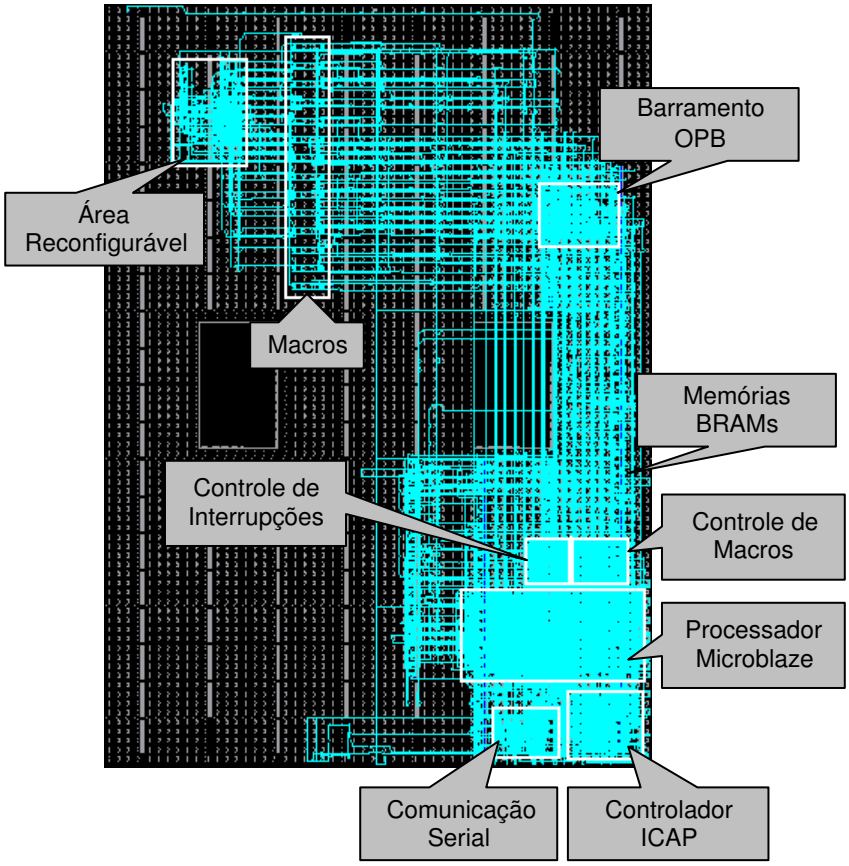


Figura 6.6: Leiaute resultante da síntese do projeto Contador Up-Down, conforme visualizado com a ferramenta FPGA Editor.

regressivo), fato que simplifica o controle de configurações, não representando um cenário que exija um controle de configurações mais complexo. Neste caso específico, o escalonador de configurações do RSCM-S é representado pela estrutura de uma fila, com apenas uma posição preenchida. Por outro lado, o estudo de caso serviu para validar a comunicação através do uso de macros e o suporte da arquitetura para atender requisições de interrupção.

A Figura 6.7 apresenta as mensagens enviadas ao PC durante o tratamento da interrupção, onde o RSCM identifica a configuração geradora e reconfigura o dispositivo com a configuração do contador regressivo. Após a reconfiguração, verifica-se através do terminal no PC que o processo de contagem ocorre regressivamente. Os dados obtidos na Figura 6.7 refletem a casualidade de uso dos mesmos flip-flops e LUTs em ambas configurações parciais, pois não existe salvamento de contexto durante o processo de reconfiguração.

Tabela 6.3: Utilização dos recursos do FPGA.

| Módulo | Número de Slices | Número de DFFs | Percentual de Slices |
|---|---|---|---|
| Microblaze | 571 | 366 | 4,17% |
| MB_OPB | 89 | 11 | 0,65% |
| OPB_hwicap | 151 | 155 | 1,1% |
| OPB_intc | 70 | 77 | 0,51% |
| DLMB | 2 | 1 | 0,01% |
| DLMB_Cntlr | 3 | 1 | 0,02% |
| ILMB | 2 | 1 | 0,01% |
| ILMB_Cntlr | 1 | 1 | 0,007% |
| RS-232 | 63 | 60 | 0,45% |
| Módulo Reconfigurável. | 90 | 135 | 0,66% |
| Módulo Controle | 99 | 136 | 0,72% |
| Área de Controle | 1051 | 809 | 7,67% |
| Área Total | 1141 | 944 | 8,33% |

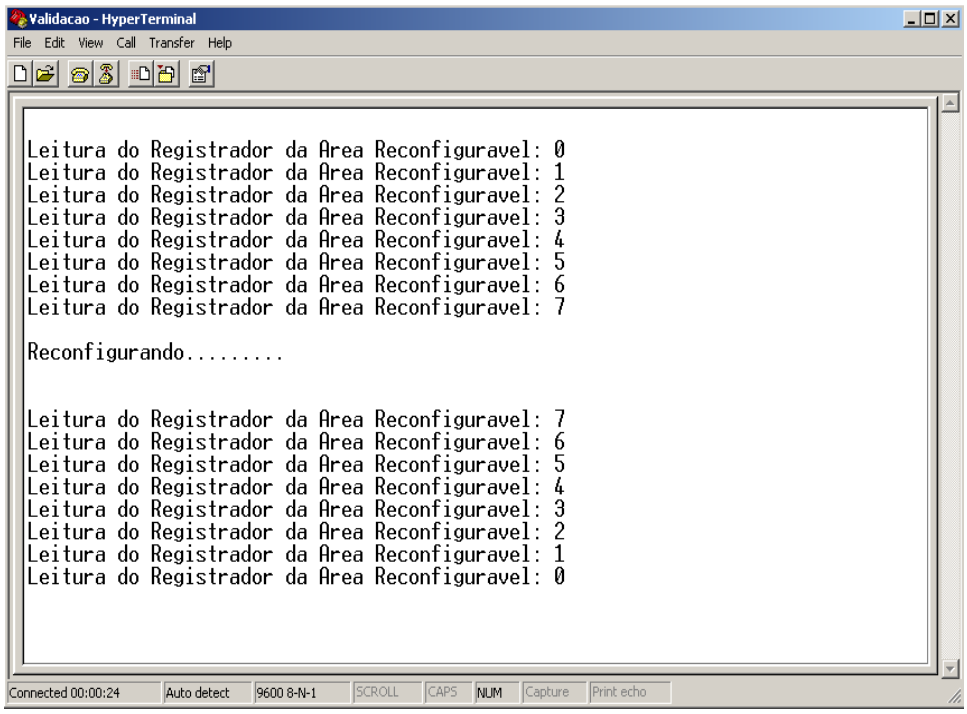


Figura 6.7: Captura de tela que comprova a contagem do módulo reconfigurável configurado como contador.

### 6.1.3 Estudo de Caso 3: Projeto Duas Áreas

O projeto Duas Áreas tem como objetivo aplicar o fluxo de projeto proposto para desenvolver SDRs mais complexos, incluindo implementações com duas áreas reconfiguráveis. Estes SDRs exigem um controle de configurações mais eficiente, com capacidade de escalonar configurações entre diferentes áreas reconfiguráveis, situação não encontrada nos estudos de caso anteriores. Visa-se aqui estabelecer uma infra-estrutura de projeto que permita a construção de SDRs escaláveis, ou seja, com duas ou mais áreas reconfiguráveis.

A arquitetura utilizada no projeto duas áreas possui estrutura de controle similar aos estudos de caso anteriores contendo, contudo, duas áreas reconfiguráveis. Neste estudo de caso também foram implementadas sobre as áreas reconfiguráveis portas lógicas de duas entradas. As portas configuradas foram *xor*, *or* e *and*. Um diagrama em blocos da arquitetura é apresentado na Figura 6.8.

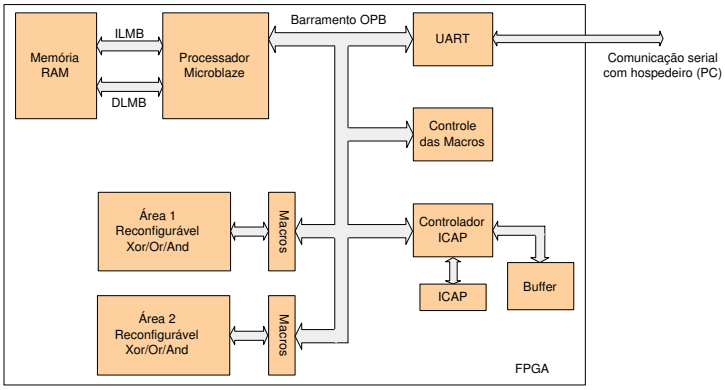


Figura 6.8: Diagrama de Blocos do projeto Duas Áreas.

Devido à estrutura da arquitetura de configuração em dispositivos Xilinx ser orientada a colunas de recursos, já foi mencionado que a forma mais adequada para definir áreas fixas e reconfiguráveis é alocar para cada uma destas um conjunto contíguo de colunas de recursos. Assim, normalmente se coloca a área fixa em um dos lados do dispositivo e ao lado desta se posiciona as áreas reconfiguráveis uma ao lado da outra. Aqui, optou-se por numerar as áreas reconfiguráveis em ordem crescente, da esquerda para a direita, e colocar a área fixa na extrema direita do dispositivo.

Assim, o módulo Controle de Macros neste projeto possui sinais independentes para macros da Área 1 e para macros da Área 2. Como a Área 1 não compartilha nenhum recurso com a Área 2, o Controle de Macros permite a operação da Área 2, bem como a comunicação desta com a

área fixa, *durante* a reconfiguração da Área 1. O mesmo não pode ocorrer na Área 2 em relação a Área 1, pois ao reconfigurar a Área 2, estar-se-á reconfigurando colunas por onde passam fios de comunicação entre a área fixa e a Área 1. Note-se que o fabricante menciona que a reconfiguração de áreas com hardware idêntico não causa efeitos transitórios. Assim, se é possível garantir que as duas configurações da Área 2 utilizam exatamente os mesmos caminhos para rotear os fios de comunicação entre a área fixa e a Área 1, a Área 1 poderia continuar operando, inclusive se comunicando com a área fixa durante a reconfiguração da Área 2. A restrição de não fazê-lo foi adotada aqui por segurança, dado não se ter certeza que, em todos os casos, o roteamento das configurações destinadas à Área 2 atenda a restrição acima. Assim, durante a reconfiguração da Área 2 a operação da Área 1 e sua comunicação com a Área fixa será sempre suspensa.

Tabela 6.4: Definição de codificação para as configurações utilizadas no projeto Duas Áreas.

| Número | Configuração |
|---|---|
| 0 | xor |
| 1 | or |
| 2 | and |

Durante o projeto do sistema, foi definida a política de reconfigurações, segundo as necessidades da aplicação. Esta política é empregada no sistema através do preenchimento da tabela TDD. Como exemplo, as configurações utilizadas foram numeradas aleatoriamente, como mostrado na Tabela 6.4 e seu fluxo de execução, representado através de um grafo, como mostrado na Figura 6.9.

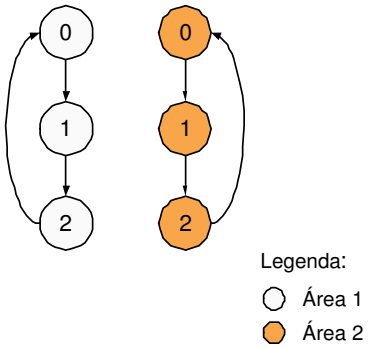


Figura 6.9: Grafo correspondente ao fluxo de execução das configurações do projeto Duas Áreas.

A política de reconfiguração representa os requisitos da aplicação para a qual o sistema será projetado. A partir desta, é possível preencher manualmente o conteúdo da tabela TDD para

que seja estabelecido o fluxo de execução do sistema. O preenchimento da tabela TDD de acordo com o exemplo é mostrado na Tabela 6.5.

Tabela 6.5: Definição da política de escalonamento de configurações para o projeto Duas Áreas.

| Config. | Npredec. | Suc. 1 |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 2 |
| 2 | 1 | 0 |

Inicialmente, as áreas reconfiguráveis são carregadas com uma porta lógica xor e, de acordo com o término de execução da configuração pela aplicação, o controlador RSCM-S aplica a política de escalonamento imposta pela TDD. Como a arquitetura do sistema não possui suporte à interrupção, o controlador de configurações RSCM-S utiliza o monitor de reconfiguração internamente à aplicação. O software do sistema foi projetado apenas para realizar operações de escrita e leitura do registrador de cada área reconfigurável, possibilitando deste modo a validação do funcionamento das portas lógicas configuradas sobre as áreas respectivas. O algoritmo usado no projeto do software do sistema é mostrado a seguir.

1. Inicializa o controle da ICAP;
2. Inicializa TDD e TAR do RSCM-S;
3. Escreve dado nas áreas reconfiguráveis;
4. Lê dados das áreas reconfiguráveis;
5. Dispara execução do módulo MR, parametrizado com a área que concluiu o processamento;
6. Voltar ao passo 3;

O software da aplicação do sistema após obter os dados processados pelas áreas reconfiguráveis determina através do MR o instante no qual um módulo deve ser reconfigurado. A identificação da configuração é enviada pelo monitor ao controlador de configurações, para que este consulte a TDD e determine a próxima configuração a ser executada. Decidida a próxima configuração, o controlador reconfigura a área livre e encerra o seu processamento, liberando o processador para execução de seu fluxo normal (escrita e leitura do registrador da área reconfigurável).

#### 6.1.3.1 Resultados

Concluída a síntese do hardware e implementação do software do projeto foram obtidos os resultados que validam o fluxo de projeto para esta implementação. O resultado da síntese do hardware é apresentado na Figura 6.10, onde é visualizado o posicionamento e roteamento dos módulos do sistema sobre o dispositivo XC2VP30.

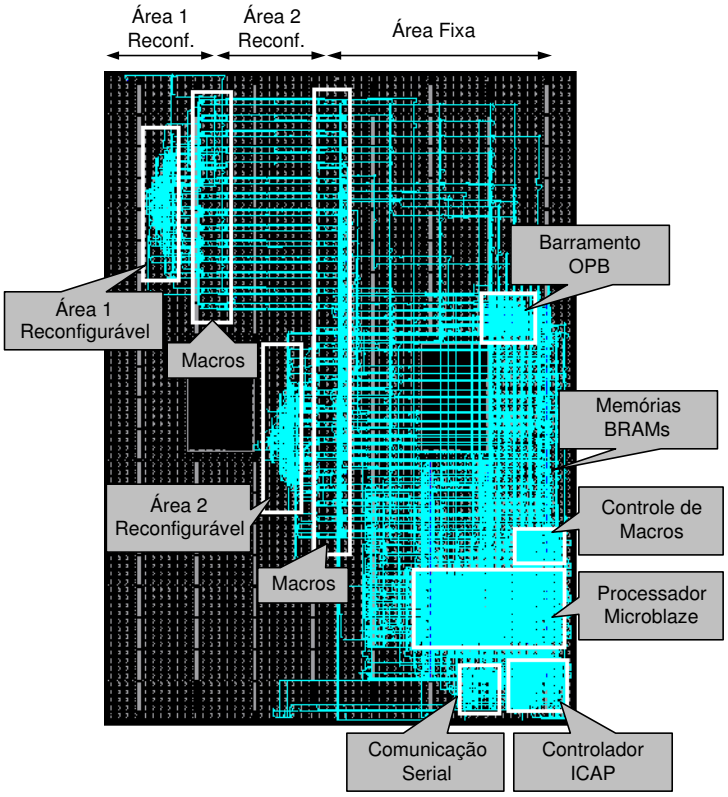


Figura 6.10: Utilização dos recursos do FPGA para implementar o projeto Duas Áreas sobre o dispositivo XC2VP30.

Nesta Figura 6.10, é possível visualizar as macros isolando as áreas reconfiguráveis da arquitetura, formando as interfaces de comunicação entre os módulos e a Área de Controle do sistema. A Área 2 é delimitada pelas macros, tanto na interface com a Área 1 quanto na interface com a Área de Controle.

### 6.1.4 Estudo de Caso 4: Teste do Controlador RSCM-S

Este estudo de caso tem como objetivo testar o funcionamento do controlador de configurações desenvolvido em software ao utilizar um grafo de dependências de configurações exemplo. Como

os estudos de casos anteriores não apresentaram um número de configurações onde houvesse a necessidade de um controle complexo surgiu a necessidade de criar um sistema exemplo para avaliar o funcionamento do controlador RSCM-S.

O estudo de caso foi realizado em um computador pessoal, infra-estrutura suficiente para testar o controle de configurações realizado inteiramente em software (segundo caso do MR). O estudo de caso visa simular um sistema contendo infra-estrutura com duas áreas reconfiguráveis e 6 configurações parciais diferentes.

O sistema exemplo apresenta o grafo de dependências entre configurações conforme apresentado na Figura 6.11.

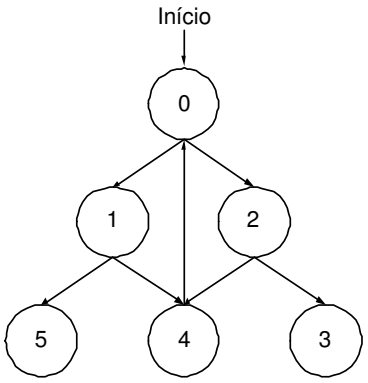


Figura 6.11: Grafo de dependências proposto para o teste do controlador de configurações.

A TDD do RSCM-S correspondente ao grafo apresentado na Figura 6.11 é mostrada na Tabela 6.6. Por convenção, os campos da TDD que não contém valores, ou seja, são considerados "*vazios*"ou "*nulos*", atribui-se o número 16 ao campo. Com esta definição pretende-se facilitar a operação do controlador.

Tabela 6.6: Preenchimento da TDD corresponde ao grafo da Figura 6.11.

| Config. | Npredec. | Suc. 1 | Suc. 2 |
|---|---|---|---|
| 0 | 1 | 1 | 2 |
| 1 | 1 | 4 | 5 |
| 2 | 1 | 3 | 4 |
| 3 | 1 | 16 | 16 |
| 4 | 2 | 0 | 16 |
| 5 | 1 | 16 | 16 |

#### 6.1.4.1 Resultados

Com a implementação do controlador e execução sobre uma suposta arquitetura com duas áreas reconfiguráveis foi possível validar seu funcionamento de maneira que possa ser empregada em um sistema embarcado auto-reconfigurável.

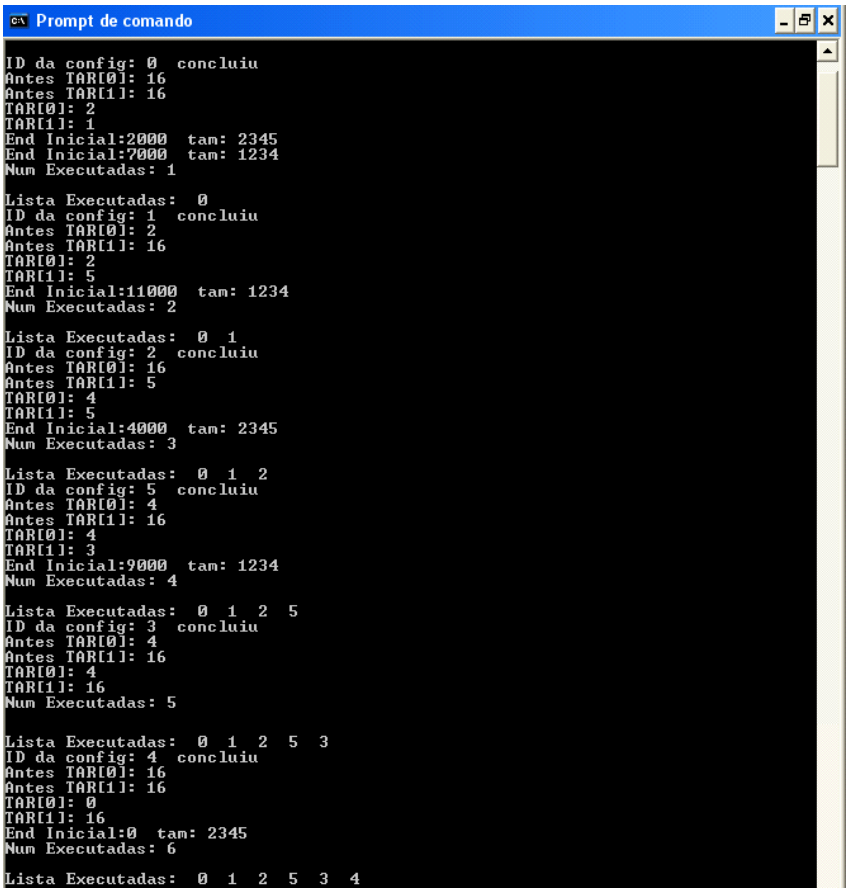


Figura 6.12: Fluxo de execução de configurações apresentadas no grafo 6.11. TAR identifica as configurações existentes em cada áreas reconfigurável.

A Figura 6.12 apresenta a captura de tela do fluxo de execução das configurações sobre a arquitetura exemplo. No estado inicial (definido segundo o grafo), o sistema possui apenas uma configuração disponível nas áreas reconfiguráveis, a configuração "*0*". Com a conclusão da configuração "*0*", o RSCM-S escalona a configuração "*1*", como indicado no grafo. Na Figura 6.12 é possível ver o endereço de memória inicial e o número de posições ocupadas pelo bitstream da configuração "*1*". Desta maneira, o sistema é teoricamente reconfigurado de forma a ter um novo estado de configurações. De acordo com a grafo apresentado, após a conclusão de cada configuração, o hardware é reconfigurado até concluir seu processamento, ou seja, a execução da

configuração ("*4*"), fato que faz o sistema retornar ao estado inicial do sistema.

## 6.2 Desenvolvimento de uma Aplicação Realista para SDRs

Com a validação das infra-estruturas para controle de configurações em software e do controlador RSCM-S através de projetos de prova de conceito, surge então a necessidade de empregar estes recursos em uma aplicação prática demonstrando a viabilidade de sistemas implementados em hardware reconfigurável como uma abordagem alternativa para desenvolvimento de SoCs.

A infra-estrutura desenvolvida dispõe de um processador capaz de atender à demanda por processamento de uma aplicação e, ao mesmo tempo, executar o controle de configurações no sistema. Neste contexto, estes recursos foram empregados para desenvolver um sistema que ofereça segurança e integridade de dados usando criptografia implementada sobre hardware reconfigurável.

O sistema oferece dois algoritmos de criptografia distintos, com o intuito de garantir segurança à dados. O software do sistema gerencia o recebimento e envio de dados e organiza-os em forma de mensagens para que sejam criptografados. Eventualmente, o algoritmo é alterado pelo sistema com o intuito de oferecer outro modo de encriptação de dados. O controlador de configurações RSCM-S tem a função de gerenciar o chaveamento entre os módulos de hardware que implementam estes algoritmos. Para implementar o sistema, foram utilizados dois IPs pré-verificados disponíveis no site OpenCores [32]: o algoritmo MD5 e o algoritmo DES56.

- **MD5**: (em inglês, *Message Digest Algorithm 5*) é usado geralmente para verificação de integridade de arquivos e como um mecanismo para validação de senhas. Este algoritmo obtém como entrada uma mensagem de comprimento arbitrário e gera uma saída criptografada com tamanho de 128 bits. O IP utilizado possui entrada e saída para mensagens com 128 bits de comprimento. Este algoritmo emprega uma função de *hash* unidirecional, ou seja, que não permite que uma mensagem seja decriptografada. Uma função *hash* recebe um valor de determinado tipo e retorna um código para ele. Na maioria dos casos, o contradomínio desta função é muito menor do que o seu domínio, ou seja, o tipo de entrada pode assumir uma gama muito maior de valores do que a função hash gera como resultado.

- **DES56**: (em inglês, *Data Encryption Standard*): é utilizado para criptografar mensagens a partir de uma chave de entrada. Este algoritmo obtém como entrada uma mensagem e uma chave, ambas com tamanho de 64 bits, e gera uma mensagem de saída também de 64

bits. A chave utilizada é constituída por 1 bit de paridade para cada byte que a compõe, de forma que o algoritmo utiliza 56 bits de dados para criptografar a mensagem de entrada.

### 6.2.1 Estrutura do Projeto

A arquitetura de hardware desenvolvida para este sistema pode ser vista na Figura 6.13.

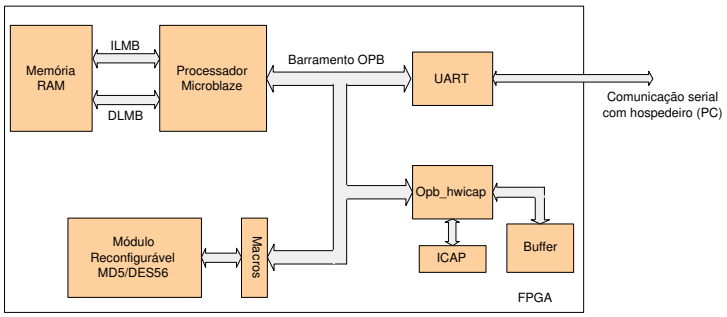


Figura 6.13: Infra-estrutura de hardware definida para o sistema de criptografia de dados.

Esta arquitetura contém basicamente os módulos utilizados nos estudos de caso anteriores, com exceção do módulo reconfigurável que implementa algoritmos de criptografia, ora MD5 ora DES56. Este sistema não utiliza comunicação por interrupção. Isto significa que o software da aplicação utiliza o monitor de reconfiguração para decidir o momento em que será alterada a configuração da área reconfigurável.

A arquitetura é inicialmente definida com a configuração do algoritmo MD5, pois este ocupa uma área maior do dispositivo [29]. Esta condição de projeto permite definir o posicionamento das macros que estabelecem a comunicação entre as áreas fixa e reconfigurável do sistema.

Segundo o fluxo de projeto, após a síntese da configuração inicial as etapas seguintes definem as configurações parciais do sistema. Neste caso, a única configuração parcial adicional será o algoritmo DES56. Esta arquitetura secundária, como definido no fluxo, é sintetizada de acordo com os arquivos .ncd e .ucf da configuração inicial de maneira a obter a definição de interface, e um posicionamento e roteamento iguais ao projeto da configuração inicial. A etapa seguinte do fluxo, é gerar os bitstreams parciais com a ferramenta CoreUnifierII- Pro [29], seguido pela compactação do arquivo de configuração parcial para que seja armazenado na memória do sistema.

### 6.2.2 Projeto de Software

Concluído o projeto de hardware do sistema, a etapa seguinte do fluxo foi definir o projeto do software. O software da aplicação foi projetado para ler um bloco de dados de entrada,

organizar o envio deste bloco em mensagens com comprimento compatível com os algoritmos implementados pelo módulo reconfigurável e obter os dados de saída, reorganizando-os em um novo bloco de dados.

Para que a comunicação seja estabelecida entre hardware e software, o projeto utiliza funções da API para leitura e escrita nos registradores do módulo reconfigurável. A aplicação também permite a monitoração das mensagens por um terminal externo, através da comunicação serial entre o sistema e um hospedeiro.

A aplicação, após realizar a encriptação do bloco de dados utiliza o monitor de reconfiguração do controlador RSCM-S para definir o instante em que o algoritmo MD5 encerra seu processamento e deve então ser substituído pelo algoritmo DES56. Neste momento, o procedimento que implementa o controlador de configurações RSCM-S é executado para reconfigurar parcialmente o sistema, alterando a área reconfigurável para conter a funcionalidade do algoritmo DES56.

Com a nova configuração, o sistema aplica a encriptação sobre o bloco de dados de entrada exibindo novamente as mensagens criptografadas no hiperterminal do hospedeiro.

### 6.2.3 Resultados

Concluído o processo de síntese das configurações inicial e parcial, foi possível verificar o posicionamento e o roteamento dos módulos das duas arquiteturas. A Figura 6.14 apresenta o resultado do processo de síntese de ambas versões do sistema.

Através da Figura 6.14, é possível verificar o mesmo posicionamento das macros em ambos os projetos, como definido.

Com o processamento da aplicação pôde-se verificar a encriptação de algumas mensagens. A Tabela 6.7 apresenta a mensagem "*mamae_ meamaPUCRS*"convertida para código hexadecimal e criptografada com ambos os algoritmos.

Tabela 6.7: Exemplo de mensagens criptografadas com ambos os algoritmos que compõem o sistema (MD5 e DES56).

| Algoritmo | Código ASCII Hexadecimal | Saída |
|---|---|---|
| MD5 | 6D616D6165206D65616D615055435253 | F72F289763D238D249FFE643E0CE01E1 |
| DES56 | 6D616D6165206D65616D615055435253 | 067BAD6B4D8333B5A02B299B19133CDD |

Os módulos de criptografia disponibilizados no site OpenCores [32] possuem arquivos de simulação que validam o funcionamento dos módulos. Com base nestes arquivos de simulação,

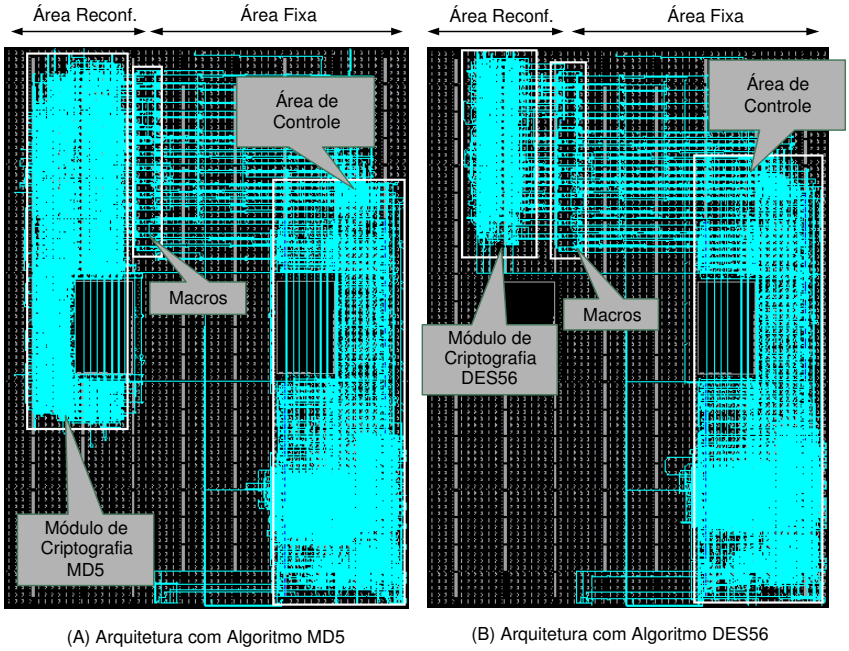


Figura 6.14: Resultado do processo de síntese das arquiteturas base (a) e secundária (b) apresentado pela Editor de FPGA.

foi realizada a validação dos projetos, com ambos algoritmos, com mesmas chaves e cadeias de caracteres utilizadas na simulação disponibilizada [32]. Foi verificado que ambas as cadeias de caracteres de saída dos algoritmos implementados no projeto são idênticas às encontradas em simulação, comprovando-se o correto funcionamento do sistema.

## 6.3 Comparação entre RSCM e RSCM-S

A infra-estrutura proposta dispõe de um processador capaz de executar o software de uma dada aplicação e gerenciar o controlador de configurações através do RSCM-S. Diferentemente, o controlador RSCM atua apenas como um sistema síncrono reativo, dependendo de sinais provenientes de uma aplicação também desenvolvida em hardware ou um processador independente do RSCM.

Para comparar estas propostas foram utilizados dois critérios: área ocupada e latência de chaveamento de configuração.

### 6.3.1 Comparação quanto ao critério área ocupada

A estrutura do controlador RSCM é dada pelo diagrama de blocos funcionais da Figura 6.15 em (A). Na Figura 6.15 (B), aparece a infra-estrutura de controle proposta neste trabalho.

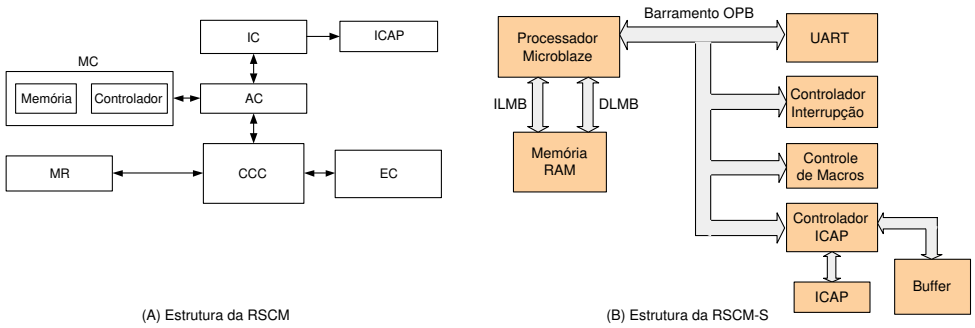


Figura 6.15: Diagrama em blocos funcionais do RSCM (A) e do RSCM-S (B).

A infra-estrutura aqui proposta possui um hardware mais complexo em relação ao controlador RSCM, pois emprega um processador de 32 bits e uma estrutura de comunicação mais complexa endereçada pelo barramento Coreconnect. A estrutura do controlador é mais simples, pois implementa apenas máquinas de estados para gerenciar o sistema.

Com a prototipação de ambas as infra-estruturas sobre o dispositivo XC2VP30 foi possível comparar a área consumida por ambos controladores. A Tabela 6.8 mostra a utilização de recursos do FPGA para implementar o RSCM.

Tabela 6.8: Utilização dos recursos do FPGA para implementar o RSCM.

| Módulo | Número de Slices | Número de FFs | Percentual de Slices |
|---|---|---|---|
| MR | 11 | 20 | 0,08% |
| EC | 42 | 69 | 0,31% |
| CCC | 71 | 97 | 0,52% |
| AC | 130 | 236 | 0,95% |
| MC | 216 | 329 | 1,58% |
| Total | 470 | 751 | 3,44% |

Comparando a utilização de recursos de ambas as estruturas de controle, é visível que a infra-estrutura proposta é sensivelmente maior que a proposta de Carvalho. A vantagem de utilizar esta infra-estrutura é o fato desta permitir o processamento de uma aplicação em software, apresentado maior flexibilidade com relação à sistemas implementados integralmente em hardware. Obviamente, acrescentar um processador de 32 bits apenas para controlar o processo

de configuração do hardware não se justifica. Contudo, hoje é praticamente impensável imaginar SoCs que não contenham um processador programável associado a memórias. Assumindo-se a existência deste processador (e.g. um MicroBlaze ou um PowerPC), e assumindo que este possui recursos disponíveis para realizar o controle de configurações, nota-se que a área adicionada a este para implementar o RSCM-S, estimada a partir da Tabela 6.3, contabilizando os módulos usados apenas para habilitar reconfiguração do hardware (OPB_hwicap e Controle de Macros) chega-se a 1,82% do FPGA, contra 3,44% de sobrecarga gerada pelo RSCM.

### 6.3.2 Comparação quanto ao critério latência de chaveamento

Em sistemas reconfiguráveis, a latência de reconfiguração é um fator crítico, que deve estar previsto no projeto, contribuindo de forma importante para a viabilidade de implementação do sistema com esta abordagem.

Conforme os estudos de caso realizados anteriormente, foi possível medir os tempos relativos ao processo de reconfiguração do sistema. Segundo as medições e análises de formas de onda, foi possível estabelecer a Equação 6.1 para determinar a latência de chaveamento de configurações parciais.

$$t_{reconf} = (t_{carga\_buffer} + t_{escrita}) \times \frac{num\_palavras}{512} \tag{6.1}$$

onde $t_{reconf}$ é a latência de reconfiguração, $t_{carga\_buffer}$ é o tempo para carregar 512 palavras de 32 bits da memória para o buffer do controlador ICAP, $t_{escrita}$ é o tempo para escrever 512 palavras do buffer no ICAP e *num_palavras* é o número total de palavras que constituem o bitstream parcial.

Segundo as medições realizadas com o analisador lógico foi possível obter os seguintes valores descritos pela Tabela 6.9:

Tabela 6.9: Medição de tempos para realizar a leitura e escrita de 512 palavras de 32 bits, utilizando freqüência de relógio de 25 MHz.

| Medição | Número de LUTs |
|---|---|
| $t_{escrita}$ | 81,88$\mu$s |
| $t_{carga\_buffer}$ s/ compac | 1,293ms |
| $t_{carga\_buffer}$ c/ compac | 1,378ms |

Tomando como exemplo o bitstream parcial do estudo de caso Contador Up-Down com

tamanho de 63673 palavras é possível calcular o tempo de reconfiguração com a equação 6.1, sabendo que a freqüência do relógio usada foi 25MHz.

$$t_{reconf} = (81,88\mu s + 1,378ms) \times \frac{63673}{512}$$

$$t_{reconf} = 181,55ms$$

Supondo que um bitstream parcial com mesmo tamanho fosse reconfigurado com o RSCM, a Equação 6.2 abaixo seria utilizada, segundo [7].

$$T_{reconf} = T_{init} + (Num_{palavras} \times T_{palavra}) \tag{6.2}$$

$$T_{reconf} = 884\eta s + (63673 \times 748\eta s)$$

$$T_{reconf} = 47,63ms$$

Com os resultados obtidos é possível comprovar que o controle em hardware é aproximadamente 4 vezes mais rápido em relação ao controle em software. Obviamente, a infra-estrutura proposta oferece aqui flexibilidade de implementação adicionalmente muito maior que o RSCM, além de oferecer maior capacidade de armazenamento de bitstreams através do emprego de compactação de bitstreams.

Uma comparação qualitativa visando mostrar características de ambos controladores com suas vantagens e inconvenientes foi realizada e apresentada na Tabela 6.10.

Tabela 6.10: Quadro com vantagens e inconvenientes relativos e características dos controladores RSCM e RSCM-S.

| **Característica Avaliada** | **RSCM** | **RSCM-S** |
|---|---|---|
| Velocidade de Configuração | Alta | Baixa (Bufferização, rotinas de leitura/escrita) |
| Auto-configuração | Não para VirtexII<br>Sim para VirtexII - Pro | Sim Via API<br>disponibilizada pelo fabricante |
| Área | Pequena se comparada a processador<br>senão grande | grande se processador considerado,<br>pequena se processador pré-existe |
| Estabilidade Funcional | Baixa, Complexa,<br>e Área Adicional | Alta, simples reescrita de software<br>para e.g. escalonamento, preempção, etc. |
| Compactação/Descompactação | Não suportada | Suportada (algoritmo Trivial) |

# Capítulo 7

# Considerações Finais

## 7.1 Resumo das Contribuições

A principal contribuição desse trabalho é a proposta e implementação de uma infra-estrutura para controle de configurações em software para hardware reconfigurável. A presença de um processador garante uma flexibilidade maior para adição de funcionalidades a SDRs, permitindo que funções que exijam processamento mais complexo (escalonamento de configurações, controle de preempção).

O trabalho realizado contribuiu para o domínio da tecnologia de FPGAs Xilinx capazes de habilitar o desenvolvimento de aplicações auto-reconfiguráveis. Este domínio possibilita a redução do uso de dispositivos de hardware externos para realizar o processo de configuração do sistema.

O presente trabalho também contribuiu com o desenvolvimento de metodologia de projetos para SDRs utilizando os ambientes do próprio fabricante Xilinx. Além disso, disponibiliza uma interface complexa para comunicação entre módulos fixos e reconfiguráveis com base nas macros propostas por Möller [29]. Esta interface foi parte fundamental para o desenvolvimento de configurações parciais em SDRs.

O trabalho também oportunizou o estudo sobre infra-estruturas que empregam reconfiguração dinâmica e parcial, bem como metodologias utilizadas para o controle de configurações. Estas infra-estruturas e metodologias forneceram o embasamento teórico necessário para melhor entender a tarefa de controle de configurações em SDRs e o desenvolvimento deste trabalho.

## 7.2 Conclusão

Neste trabalho, pôde-se evidenciar a falta de ferramentas para o projeto e suporte à implementação de SDRs. Embora diversos estudos tenham sido realizados enfatizando as vantagens do uso de SDRs, fabricantes ainda não disponibilizam ferramentas e métodos que habilitem a geração automática de sistemas dotados de capacidade de reconfiguração dinâmica e parcial. O fluxo de projeto convencional para desenvolvimento de hardware não é adequado ao projeto de SDRs, sendo necessário utilizar operações manuais custosas em termos de tempo para conceber sistemas desta natureza.

O emprego destes sistemas exige obrigatoriamente um subsistema que gerencie as configurações existentes em um SDR. A proposta de infra-estrutura e controle de configurações em software (RSCM-S) surge como uma alternativa prática eficaz para o controle de configurações, bem como um meio que suprir as necessidades de SDRs no que diz respeito ao emprego da flexibilidade de software para controle do sistema.

A infra-estrutura proposta e implementada aqui teve como objetivo fazer evoluir a proposta RSCM de Carvalho [7] oferecendo a possibilidade de auto-reconfiguração sem necessidade de dispositivos externos ao FPGA. O controle de configurações através de software proporciona uma maior flexibilidade de implementação em relação ao RSCM integralmente implementada em hardware. Porém, a estrutura torna-se complexa exigindo maior área (se considerar a área do processador) e apresentando latência de reconfiguração muito maior em relação ao RSCM. Projetistas que desejam desenvolver SDRs devem raciocinar sobre o compromisso entre flexibilidade e desempenho do sistema, observando as vantagens e inconvenientes destas duas propostas.

O RSCM-S possui recursos para armazenamento de bitstreams parciais compactadas, e inclui a capacidade de descompactação automatizada de bitstreams e escalonamento estático de configurações. Os estudos de caso realizados no Capítulo 6 comprovam seu funcionamento, tendo-se assim atingido este objetivo do trabalho.

## 7.3 Trabalhos Futuros

O desenvolvimento do RSCM-S pode e deve evoluir. A infra-estrutura possui uma restrição para armazenar bitstreams parciais, mesmo empregando compactação de bitstreams. As plataformas de prototipação disponibilizam memória suficiente para armazenar múltiplos bitstream, porém os controladores de acesso à memória disponíveis no ambiente EDK não foram habilita-

dos a serem empregados junto com a versão atual do RSCM-S. Uma possibilidade é desenvolver estudos a fim de dominar esta tecnologia para que se habilite o uso de capacidades muito maiores de armazenamento de bitstreams para estes sistemas.

A infra-estrutura proposta apresentou uma latência de chaveamento superior à latência encontrada na proposta de Carvalho [7]. Isto ocorre principalmente pela restrição encontrada no controlador da ICAP disponível no ambiente EDK, que utiliza um dispositivo de armazenamento temporário com pequena capacidade para escrita e leitura de configurações do dispositivo. Estudos sobre o desenvolvimento de um controlador mais eficiente seria uma alternativa para reduzir a latência de chaveamento, tanto para esta proposta quanto para o RSCM, que não dispõe de tal recurso.

O ambiente de desenvolvimento EDK permite projetar arquiteturas de hardware complexas, abstraindo detalhes de implementação como interconectar módulos com interfaces contendo dezenas ou centenas de fios. Por outro lado, este ambiente não possui flexibilidade para permitir otimizar algumas estruturas, como é o caso do controlador da ICAP. Com esta possibilidade sendo aberta pelo fabricante, seria possível desenvolver arquiteturas de controle de configuração em software mais otimizadas.

Uma importante etapa do fluxo de desenvolvimento de SDRs é a definição de sua planta baixa. Nesta etapa, as interfaces entre áreas fixas e reconfiguráveis existente nas arquiteturas base e secundárias devem ser obrigatoriamente as mesmas. A verificação destas interfaces é realizada hoje obrigatoriamente de forma visual com a ferramenta FPGA Editor. Esta etapa de verificação poderia ser automatizada com a construção de uma ferramenta que interpretasse os arquivos de configurações gerados. Isto reduziria consideravelmente o tempo de projeto de SDRs.

# Referências Bibliográficas


[1] Algotronix, Ltd. *Configurable Array Logic 1024*, April 1989. Data Sheet - 41 pages.

[2] L. Antoni, R. Leveugle, and B. Feher. Using run-time reconfiguration for fault injection applications. *IEEE Transactions on Instrumentation and Measurement*, 52(5):1468–1473, October 2003.

[3] Atmel Corp. *AT40K Series Configuration*, 2000. Capturado em http://www.atmel.com/atmel/postscript/doc1009.ps.zip.

[4] Atmel Corp. *Implementing Cache Logic with FPGAs*, September 2000. Application Note - 5 pages.

[5] J. Batlle, J. Martí, P. Ridao, and J. Amat. A new FPGA/DSP-based parallel architecture for real-time image processing. *Real-Time Imaging*, 8(5):345–356, October 2002.

[6] E. Brião. Reconfiguração parcial e dinâmica para núcleos de propriedade intelectual. Master's thesis, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, Brazil, 2003. (In Portuguese - 104 Pages).

[7] E. Carvalho. RSCM - controlador de configuração para sistemas de hardware reconfigurável. Master's thesis, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, Brazil, 2004. (In Portuguese - 122 pages).

[8] E. Carvalho, N. Calazans, E. Brião, and F. Moraes. PaDReH - a framework for the design and implementation of dynamically and partially reconfigurable systems. In *Symposium on Integrated Circuits and Systems Design - SBCCI*, pages 10–15, Pernambuco, Brazil, September 2004.

[9] J. Chen, J. Moon, and K. Bazargan. A reconfigurable FPGA-based readback signal generator for hard-drive read channel simulator. In *ACM/IEEE Design Automation Conference - DAC*, pages 349–354, New Orleans, USA, June 2002.

[10] A. Dawood, S. Visser, and J. Williams. Reconfigurable FPGAs for real time image processing in space. In *14th International Conference on Digital Signal Processing - ICDSP*, pages 845–848, Santorini, Greece, July 2002.

[11] A. Dehon. DPGA utilization and application. In *4th International Symposium on Field Programmable Gate Arrays - FPGA*, pages 115–121, Monterey, USA, February 1996.

[12] A. Dehon. Comparing computing machines. In *Configurable Computing: Technology and Applications, Proc. SPIE 3526*, pages 124–133, Massachusetts, USA, November 1998.

[13] A. Dehon. The density advantage of configurable computing. *IEEE Computer*, 33(4):41–49, April 2000.

[14] B. Griese, E. Vonnahme, M. Porrmann, and U. Rückert. Hardware support for dynamic reconfiguration in reconfigurable SoC architectures. In *14th International Conference on Field Programmable Logic and Applications - FPL*, pages 842–846, Antwerp, Belgium, September 2004. Springer-Verlag Berlin Heidelberg.

[15] R. Gupta and Y. Zorian. Introducing core-based system design. *IEEE Design and Test of Computers*, 14(4):15–25, October 1997.

[16] R. Hartenstein, M. Herz, T. Hoffmann, and U. Nageldinger. Kress Array Xplorer: a new CAD environment to optimize reconfigurable datapath array architectures. In *Asia and South Pacific Design Automation Conference - ASP-DAC*, pages 163–168, Yokohama, Japan, January 2000. ACM.

[17] J. Henkel. Closing the SoC design gap. *Computer*, 36(9):119–121, September 2003.

[18] M. Huebner, T. Becker, and J. Becker. Real-time LUT-based network topologies for dynamic and partial FPGA self-reconfiguring. In *Symposium on Integrated Circuits and Systems Design - SBCCI*, pages 28–32, Pernambuco, Brazil, September 2004.

[19] M. Huebner, M. Ullmann, L. Braun, A. Klausmann, and J. Becker. Scalable application-dependent network on chip adaptivity for dynamical reconfigurable real-time systems. In

*14th International Conference on Field Programmable Logic and Applications - FPL*, pages 1037–1041, Antwerp, Belgium, September 2004. Springer-Verlag Berlin Heidelberg.

[20] IBM Corporation. *Device Control Register Bus Toolkit - v2.6 Edition*, July 2000. User's Manual - 50 pages.

[21] IBM Corporation. *On-chip Peripheral Bus Functional Model Toolkit - V3.5 Edition*, June 2003. User's Manual - 80 pages.

[22] IBM Corporation. *Processor Local Bus Functional Model Toolkit - v4.9.2 Edition*, June 2003. User's Manual - 110 pages.

[23] Heinz Nixdorf Institute. RAPTOR2000 - A Modular Rapid-Prototyping System. Captured in http://www.hni.uni-paderborn.de/sct/raptor/index_e.php3. Last access: July 2005.

[24] R. Kozma, C. Horvath, G. Hosszu, and F. Kovacs. Atmel fpga and altera fpslic realization of a fast pattern recognition algorithm. In *International Symposium on Signals, Circuits and Systems - ISSCS*, pages 41–44, Iasi, Romania, July 2003.

[25] P. Lysaght. Future design tools for plataform FPGAs. In *Symposium on Integrated Circuits and Systems Design - SBCCI*, pages 275–280, São Paulo, Brazil, September 2003.

[26] Memec, Inc. *Virtex-II Pro$^{TM}$ FF1152 Development Board User's Guides*, v1.2 edition, April 2004. User Guide - 51 pages.

[27] D. Mesquita. Contribuições para reconfiguração parcial, remota e dinâmica de FPGAs. Master's thesis, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, Brazil, 2002. (In Portuguese).

[28] J. Mignolet, V. Nollet, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins. Infrastructure for design and management of relocatable tasks in a heterogeneous reconfigurable system-on-chip. In *6th Design, Automation and Test in Europe Conference and Exhibition - DATE*, pages 986–991, Munich, Germany, March 2003.

[29] L. Möller. Sistemas dinamicamente reconfiguráveis com comunicação via redes intra-chip. Seminário de Andamento - 14 pages, July 2005.

[30] National Semiconductor Corp. National complementary gallium arsenide configurable logic array. *Navigator Magazine*, 14:1, July 1998. 1 page.

[31] V. Nollet, T. Marescaux, and D. Verkest. Operating-system controlled network on chip. In *ACM/IEEE Design Automation Conference - DAC*, pages 256–259, San Diego, USA, June 2004.

[32] OpenCores Org. Crypto core. Captured in http://www.opencores.org/browse.cgi/filter /category_crypto. Last access: September 2005.

[33] L. Ost. Redes intra-chip parametrizáveis com interface padrão ocp para síntese em hardware. Master's thesis, Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, Brazil, 2003. (In Portuguese - 136 pages).

[34] I. Page. Reconfigurable processor archictetures. *Microprocessors and Microsystems*, 20(5):185–196, May 1996.

[35] J. Palma, A. Mello, L. Möller, F. Moraes, and N. Calazans. Core communication interface for FPGAs. In *Symposium on Integrated Circuits and Systems Design - SBCCI*, pages 183–188, São Paulo, Brazil, September 2002.

[36] B. Radunovic and V. Milutinovic. A survey of reconfigurable computing architectures. In *8th International Workshop on Field Programmable Logic and Applications FPL*, pages 376–385, Talling, Estônia, September 1998. Springer-Verlag Berlin Heidelberg.

[37] J. Resano, D. Mozos, and F. Catthoor. A hybrid prefetch scheduling heuristic to minimize at run-time the reconfiguration overhead of dynamically reconfigurable hardware. In *8th Design, Automation and Test in Europe Conference and Exhibition - DATE*, pages 106–111, Munich, Germany, March 2005.

[38] J. Resano, D. Mozos, D. Verkest, F. Catthoor, and S. Vernalde. Specific scheduling support to minimize the reconfiguration overhead of dynamically reconfigurable hardware. In *ACM/IEEE Design Automation Conference - DAC*, pages 119–121, San Diego, USA, June 2004.

[39] J. Resano, D. Verkest, D. Mozos, S. Vernalde, and F. Catthoor. A hybrid design-time/run-scheduling flow to minimise the reconfiguration overhead of FPGAs. *Microprocessors and Microsystems*, 28(5):291–301, 2004.

[40] E. Sanchez, M Sipper, J.-O. Haenni, J.-L. Beuchat, A. Stauffer, and A. Perez-Uribe. Static and dynamic configurable systems. *IEEE Transactions on Computers*, 48(6):556–564, June 1999.

[41] R. Schaller. Moore's law: past, present and future. *IEEE Spectrum*, 34(6):52–59, June 1997.

[42] I. Skliarova and A. Ferrari. The design and implementation of a reconfigurable processor for problems of combinatorial computation. *Journal of Systems Architecture*, 49(4-6):211–226, September 2003.

[43] R. Soares. Aplicações práticas implementadas com sistemas de hardware reconfigurável em tempo de execução. Trabalho Individual I (PPGCC - Pontifícia Universidade Católica do Rio Grande do Sul) - 42 pages, May 2004.

[44] R. Soares. Sistemas computacionais auto-reconfiguráveis e aplicações em visão computacional. Trabalho Individual II (Instituição - Pontifícia Universidade Católica do Rio Grande do Sul) - 39 pages, August 2004.

[45] C. Torres-Huitzil and M. Arias-Estrada. Reconfigurable vision system for real-time applications. In *SPIE Electronic Imaging 2002 - Photonics West*, pages 124–132. Instituto Nacional de Astrofisica, Optica, y Electronica, Nasser Kehtarnavaz, March 2002.

[46] M. Ullmann, M. Huebner, B. Grimm, and J. Becker. An FPGA run-time system for dynamical on-demand reconfiguration. In *18th International Parallel and Distributed Processing Symposium*, pages 135–142, Santa Fe, USA, April 2004.

[47] J. Williams and N. Bergmann. Embedded linux as a platform for dynamically self-reconfiguring systems-on-chip. In *The International Conference on Engineering of Reconfigurable Systems and Algorithms - ERSA*, pages 163–169, Las Vegas, USA, June 2004.

[48] M. Wirthlin and B. Hutchings. Improving funcional density using run-time circuit reconfiguration. *IEEE Transactions on VLSI Systems*, 6(2):247–256, June 1997.

[49] Xilinx, Inc. *XC6200 Field Programmable Gate Arrays*, v1.1 edition, April 1997. Product Description.

[50] Xilinx, Inc. *Virtex-II 1.5V Field-Programmable Gate Arrays*, Data Sheet v1.7 edition, October 2001. 7 pages.

[51] Xilinx, Inc. *Virtex-II Plataform FPGA*, User Guide v1.5 edition, December 2002. 460 pages.

[52] Xilinx, Inc. *Getting Started with EDK 6.2*, December 2003. Reference Guide - 10 pages.

[53] Xilinx, Inc. *Synthesis and Verification Design Guide*, Design Guide v4.0 edition, June 2003. 300 pages.

[54] Xilinx, Inc. *User Core Templates Reference Guide*, Reference Guide v1.1 edition, June 2003. 56 pages.

[55] Xilinx, Inc. *Device Drivers Documentation*, Reference Guide v1.0 edition, January 2004. 1305 pages.

[56] Xilinx, Inc. *Dynamic Reconfiguration of RocketIO MGT Attributes*, v2.2 edition, February 2004. Application Note 660 - 9 pages.

[57] Xilinx, Inc. *In-Circuit Partial Reconfiguration of RocketIO Attributes*, v2.4 edition, May 2004. Application Note 662 - 29 pages.

[58] Xilinx, Inc. *MicroBlaze Processor Reference Guide*, Reference Guide v4.0 edition, August 2004. 132 pages.

[59] Xilinx, Inc. *RocketIO Transceiver Bit-Error Rate Tester*, v2.0.2 edition, May 2004. Application Note 661 - 32 pages.

[60] Xilinx, Inc. *Two Flows for Partial Reconfiguration: Module Based or Difference Based*, v1.2 edition, September 2004. Application Note 290 - 28 pages.

[61] Xilinx, Inc. *Virtex Series Configuration Architecture User Guide*, v1.7 edition, October 2004. Application Note 151 - 45 pages.

# Apêndice A

# Código do RSCM-S

```
//Bibliotecas
#include "xparameters.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <xhwicap.h>
#include <xhwicap_i.h>
#include "hard1.h"
#include "hard2.h"

//Definições para RSCM-S e Controle do ICAP
#define LIM                         512
#define XHI_TARGET_DEVICE           XHI_READ_DEVICEID_FROM_ICAP
#define XHI_EX_BITSTREAM_LENGTH     5
#define XHI_EX_ONE_WORD             1
#define XHI_EX_ADDRESS_OFFSET       0

#define CONFIG  6   // número de configurações
#define FATIAS  2   // número de áreas configuráveis (fatias)
#define LIVRE   16  // identificador para área livre
#define NULO    15  // identificador para valor nulo (não existente)
```

```
//variaveis ICAP
XHwIcap MyIcap;
XAssertCallback MyCallBack;

//Função de Controle
void XHwIcap_AssertHandler (char* FilenamePtr, int LineNumber)
{ printf("Assert occured: %s at %d \n\r",FilenamePtr,LineNumber);
  return;}

int estado;

//Função pra identificar padrões de compactação
int verifica(int valor)
{
   int u;
   int desloc;

   desloc = valor >> 16;
   if (desloc == 0x1234)
   {
       estado = 1;
       u = valor & 0xFFFF;
   }
   else
       if (desloc == 0x4321)
       {
          estado = 2;
          u = valor & 0xFFFF;
       }
       else
           {
              estado = 0;
```

```
              u = 0;
           }
return u;}

//Estruturas de controle do RSCM-S
struct tabela{  int n_predec;
                int suc1;
                int suc2;} tdd[CONFIG];
struct ptr{ int config;
            int fatia;} prox_config[CONFIG];
struct map{ int inicio;
            int tam; } mapa[FATIAS][CONFIG];

//vetor: equivalente ao num de fatias disponíveis armazena as configs. alocadas
int tar[FATIAS];
// fila de configs escalonadas pelo EC
int fila_escalonadas[CONFIG];
// configs já executadas
int lista_executadas[CONFIG];
//backup do número de predecessoras das configs
int bck[CONFIG];

//variaveis globais de controle do RSCM-S
int pos_lista = 0;
int pos_fila = 0;

/*-------------------------------------------------|
|---------------Controlador RSCMS-------------------|
|-------------------------------------------------*/
void rscm_s(int id_config)
{
   int sucessoras[FATIAS];
```

```
   int n=0,k,u,i,Status,pos=0;
   int ja_executada,cont1=0;
   int inicio,tam;

/*-------------------------------------------------|
|-------------------Monitor-----------------------|
|-------------------------------------------------*/
   printf("\n\r\n\nID da config: %d  concluiu",id_config);
/*-------------------------------------------------*/

/*-------------------------------------------------|
|----------Controle Central de Configurações----------|
|-------------------------------------------------*/
/*--controle do CCC sobre execução das configurações--*/
//verifica se config ainda não foi executada,
if (pos_lista != 0)
{
    for(i=0;i<pos_lista;i++)
       if( lista_executadas[i] == id_config )
          ja_executada = 1;
    if(ja_executada != 1)
    {
       //coloca na fila a config executada
       lista_executadas[pos_lista] = id_config;
       //incrementa a posição da lista
       pos_lista++;
       ja_executada = 0;
    }
 }
 else
    {
       //coloca na fila a config executada
```

```
        lista_executadas[pos_lista] = id_config;
        //incrementa a posição da fila
        pos_lista++;
    }
//Restaura o hitórico de precedências da Config
tdd[id_config].n_predec = bck[id_config];

//Atualiza Tabela de Alocação de Recursos (libera a fatia onde config terminou)
for(i=0;i<FATIAS;i++)
{
   if(tar[i] == id_config)
      tar[i] = LIVRE;
   printf("\n\rAntes TAR[%d]: %d",i,tar[i]);
}
/*--------Fim do CCConfiguração----------------------*/

/*-------------------------------------------------|
|-----------Escalonador de Configurações-----------|
|-------------------------------------------------*/
/* Verifica Fila de Escalonadas e Atualiza a
fila com proximas a serem escalonadas--*/
if(tdd[id_config].suc1 != NULO) //identifica a primeira sucessora
{
   //copia para o vetor de sucessoras
   sucessoras[0] = tdd[id_config].suc1;
   //identifica e decrementa as
   if(tdd[sucessoras[0]].n_predec != 0)
   {
      //predecessoras das sucessoras
      tdd[sucessoras[0]].n_predec--;
      //verifica se a sucessora pode ser escalonada
      if(tdd[sucessoras[0]].n_predec == 0)
```

```
      {
         //coloca na fila de escalonadas
         fila_escalonadas[pos_fila] = sucessoras[0];
         //incrementa a posição da fila de escalonadas
         pos_fila++;
      }
   }
}

if(tdd[id_config].suc2 != NULO) //identifica a segunda sucessora
{
   //copia para o vetor de sucessoras
   sucessoras[1] = tdd[id_config].suc2;

   //identifica e decrementa as predecessoras das sucessoras
   if(tdd[sucessoras[1]].n_predec != 0)
   {
      tdd[sucessoras[1]].n_predec--;
      //verifica se a sucessora pode ser escalonada
      if(tdd[sucessoras[1]].n_predec == 0)
      {
         //coloca na fila de escalonadas
         fila_escalonadas[pos_fila] = sucessoras[1];
         //incrementa a posição da fila de escalonadas
         pos_fila++;
      }
   }
}
/*------------Fim do Escalonador----------------------*/

/*-------------------------------------------------|
|---------Controle Central de Configurações-----------|
```

```
|-------------------------------------------------*/
//atualiza Tabela de Alocação de Recursos
//percorre todas as fatias reconfiguráveis
for(i=0;i<FATIAS;i++)
{
   //verifica se existem fatias livres e configs na fila do escalonador
   if((tar[i] == LIVRE) && (pos_fila != 0))
   {
       //atribui config a fatia livre
       tar[i] = fila_escalonadas[cont1];
       prox_config[cont1].config = tar[i];
       prox_config[cont1].fatia = i;

       //identifica o número de configurações que devem ser reconfiguradas
       cont1++;
       //decrementa a posição da fila do escalonador.
       pos_fila--;
       for(k=0;k<pos_fila;k++) //reorganiza a fila de escalonadas
           fila_escalonadas[k]=fila_escalonadas[k+1];
   }
   printf("\n\rTAR[%d]: %d",i,tar[i]);
}
/*-----------Fim do CCConfiguração-------------------*/

/*-------------------------------------------------|
|----------------Auto-Configurador-----------------|
|-------------------------------------------------*/
for(n=0;n<cont1;n++)
{
   inicio = mapa[prox_config[n].fatia][prox_config[n].config].inicio;
   tam = mapa[prox_config[n].fatia][prox_config[n].config].tam;
```

```
printf("\nReconfigurando Config:%d",prox_config[n].config);
printf("\nEnd Inicial:%X  tam: %d",inicio,tam);

for(k = inicio; k < (inicio+tam); k++)
{
   u = verifica(memoria[k]);
   switch(estado)
   {
     case 1:
         for(i=0;i<u;i++)
         {
           XHwIcap_StorageBufferWrite(&MyIcap, pos, 0x00000000);
           pos++;
           if(pos == LIM)
           {
              Status = XHwIcap_DeviceWrite(&MyIcap, 0x00000000, LIM);
              pos=0;
           }
         }
         break;
     case 2:
         for(i=0;i<u;i++)
         {
           XHwIcap_StorageBufferWrite(&MyIcap, pos, 0x20000000);
           pos++;
           if(pos == LIM)
           {
              Status = XHwIcap_DeviceWrite(&MyIcap, 0x00000000, LIM);
              pos=0;
           }
         }
         break;
```

```
         default:
             XHwIcap_StorageBufferWrite(&MyIcap, pos, par_up[k]);
             pos++;
             if(pos == LIM)
             {
                Status = XHwIcap_DeviceWrite(&MyIcap, 0x00000000, LIM);
                pos=0;
             }
             break;
      }
    }
Status = XHwIcap_DeviceWrite(&MyIcap, 0x00000000, pos);
}//fim do contador
cont1 = 0;
/*-----------------------------------------------------|
|----FIM----Descompactador e Reconfigurador-----------|
|-----------------------------------------------------*/
}//fim do RSCMS

main() {
.
/*-----------------------------------------------------|
|---------INICIALIZAÇÃO DO CONTROLADOR ICAP-----------|
|-----------------------------------------------------*/
// Setup assert handler
MyCallBack = &XHwIcap_AssertHandler;
XAssertSetCallback(MyCallBack);

Status = XHwIcap_Initialize(&MyIcap,
         XPAR_OPB_HWICAP_0_DEVICE_ID,XHI_TARGET_DEVICE);

if (Status == XST_DEVICE_IS_STARTED) {
```

```
   print("\n\rDevice is already initialized\n\r.");
} else if (Status != XST_SUCCESS) {
       printf("\n\rFailed to initialize: %d \n\r",Status);
       exit(-1);
       }

/*-----------------------------------------------------|
|------------Mapeamento da Memória--------------------|
|-----------------------------------------------------*/
   mapa[0][0].inicio = 0x0000;
   mapa[0][0].tam = 2345;
   mapa[0][1].inicio = 0x1000;
   mapa[0][1].tam = 2345;
   mapa[0][2].inicio = 0x2000;
   mapa[0][2].tam = 2345;
   mapa[0][3].inicio = 0x3000;
   mapa[0][3].tam = 2345;
   mapa[0][4].inicio = 0x4000;
   mapa[0][4].tam = 2345;
   mapa[0][5].inicio = 0x5000;
   mapa[0][5].tam = 2345;
   mapa[1][0].inicio = 0x6000;
   mapa[1][0].tam = 1234;
   mapa[1][1].inicio = 0x7000;
   mapa[1][1].tam = 1234;
   mapa[1][2].inicio = 0x8000;
   mapa[1][2].tam = 1234;
   mapa[1][3].inicio = 0x9000;
   mapa[1][3].tam = 1234;
   mapa[1][4].inicio = 0x10000;
   mapa[1][4].tam = 1234;
   mapa[1][5].inicio = 0x11000;
```

```
   mapa[1][5].tam = 1234;

//tabela de alocação de recursos: Fatia(0) => config 0
//tabela de alocação de recursos: Fatia(1) => livre
   tar[0] = 0;
   tar[1] = LIVRE;

/*-----------------------------------------------------|
|-----Inicialização da tabela de descritores----------|
|-----------------------------------------------------*/
   bck[0] = tdd[0].n_predec = 1;
   tdd[0].suc1 = 1;
   tdd[0].suc2 = 2;
   bck[1] = tdd[1].n_predec = 1;
   tdd[1].suc1 = 4;
   tdd[1].suc2 = 5;
   bck[2] = tdd[2].n_predec = 1;
   tdd[2].suc1 = 3;
   tdd[2].suc2 = 4;
   bck[3] = tdd[3].n_predec = 1;
   tdd[3].suc1 = NULO;
   tdd[3].suc2 = NULO;
   bck[4] = tdd[4].n_predec = 2;
   tdd[4].suc1 = 0;
   tdd[4].suc2 = NULO;
   bck[5] = tdd[5].n_predec = 1;
   tdd[5].suc1 = NULO;
   tdd[5].suc2 = NULO;
/*-----Fim da Inicialização da tabela de descritores---*/
.
}
```