Universidade Federal de Pernambuco

Centro de Informática

Graduação em Ciência da Computação

ARQUITETURA PARA RECONFIGURAÇÃO ESTÁTICA E DINÂMICA DE INSTRUÇÕES NO PROCESSADOR LEON

Autor: Halmos Fernando do Nascimento
Orientador: Manoel Eusébio de Lima

Recife, 7 de Abril de 2004

Universidade Federal de Pernambuco
Centro de Informática

Halmos Fernando do Nascimento

ARQUITETURA PARA RECONFIGURAÇÃO ESTÁTICA E DINÂMICA DE INSTRUÇÕES NO PROCESSADOR LEON

Trabalho apresentado ao Programa de Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Manoel Eusébio de Lima

Recife, 7 de Abril de 2004

Aos meus pais, Hélio e Leninha, e a Milena, pelo apoio e incentivo que sempre me deram.

Agradecimentos

Agradeço a Deus, por sempre ter me dado forças nos momentos mais necessários. A meu orientador, Manoel Eusébio de Lima, pelo incentivo e confiança. A Júlio Alexandrino e Paulo Sérgio pela imensa ajuda e dedicação. A Abel, Péricle e Remy pelo apoio e todos que de uma forma ou de outra contribuíram pra a realização deste trabalho.

Halmos Fernando do Nascimento

Resumo

Este trabalho tem como objetivo o estudo de uma arquitetura dinamicamente reconfigurável para prototipação rápida de sistemas digitais baseado na CPU Leon. Assim, a partir de uma plataforma pré-estabelecida, será estudada uma arquitetura que propõe a reconfiguração dinâmica de instruções de um sistema Leon, em uma plataforma *Virtex-II*. Nesta proposta, um componente *Virtex-II* hospedará o *core* da CPU Leon e toda a sua estrutura de integração em uma área fixa da FPGA, enquanto que parte de sua estrutura de instruções pode ser dinamicamente alterada, sem que seja necessário parar o processador (reconfiguração *on-the-fly*).

Palavras-chave:

Reconfiguração Dinâmica Parcial, Leon, Virtex-II, Reconfiguração on-the-fly, Prototipação.

Abstract

This work aims the study of a dynamically reconfigurable architecture for digital systems fast prototipation, based on Leon CPU. Thus, from a previously established platform, an architecture will be studied that considers dynamic reconfiguration of instructions of a Leon system, in a Virtex-II platform. In this proposal, a Virtex-II component will house the Leon CPU core and all its structure of integration in a fixed area of the FPGA, while part of its instruction structure can be dynamically modified, without be necessary to stop the processor (on-the-fly reconfiguration).

Keywords:

Partial Dynamic Reconfiguration, Leon, Virtex-II, On-the-fly reconfiguration, Prototipation.

Sumário

Lista de Figuras

Lista de Tabelas

1 INTRODUÇÃO

Com a necessidade cada vez maior de integração de funções e velocidade na concepção de sistemas embarcados, menor *time-to-market*, torna-se necessário o uso de tecnologias cada vez mais versáteis para a prototipação rápida destes sistemas. Estas novas tecnologias dizem respeito a novas metodologias de projetos, linguagem de especificação de componentes de hardware e software e, estilos de projeto de implementação do protótipo.

Nos últimos anos, a tecnologia de dispositivos lógicos programáveis, especialmente os FPGAs *(Field Programmable Gate Array)* [Knapp; Tavana] tem evoluído bastante com um dos principais estilos de projetos para prototipação rápida de sistemas, produzindo dispositivos cada vez mais densos, com altos níveis de desempenho e menores custos de fabricação[Ribeiro; Marques]. Além disto, os fabricantes destes dispositivos, têm introduzido cada vez mais recursos de reconfigurabilidade, possibilitando projetos de sistemas tanto digitais, como analógicos, com comportamento de reconfiguração estático ou dinâmico (total ou parcial[1]).

Com o aumento na densidade de transistores e maior disponibilidade de lógica para implementação, tornou-se possível o desenvolvimento de sistemas inteiros em único componente, permitindo redução de consumo de energia, maior confiabilidade e menor custo na prototipação. O desenvolvimento de sistemas inteiros em um único chip *(System-on-Chip-SoC*), exige no entanto, uma atenção especial nos tipos de componentes, suas conexões, e ambiente de testes. Hoje em dia, *SoCs* tem sido desenvolvido baseados em plataformas previamente definidas, o que permite maior velocidade na implementação de um protótipo, com padronização de interfaces e de processadores. Algumas destas plataformas já possuem inclusive CPUs com interfaces, previamente incorporadas ao circuito de prototipação (FPGAs) [Bray].

[1] Sistemas que possuem a capacidade de alterar parcialmente a sua funcionalidade enquanto ativo, sem prejudicar o funcionamento da lógica restante que pode está em operação [Ribeiro; Marques].

No entanto, o desenvolvimento de *SoCs* requer em geral, a inclusão de componentes, além da CPU, ao sistema, para que a toda a funcionalidade da aplicação seja realizada de acordo com os requisitos do usuário. Estes componentes, denominados de *Intellectual Property Cores (IP-cores)*, podem ser representados por barramentos, memórias e dispositivos de I/O (UART, Timers, USB, Bluetooth, etc). Estes *IP-Cores* podem ainda de classificados como *hard*, *firm* ou *soft cores* [Küçükçakar][Junchao; Weiliang; Shaojun].

A adoção de plataformas reconfiguráveis para desenvolvimento de *SoCs* baseadas em CPUs integradas a IPs é uma realidade. Este trabalho visa utilizar este paradigma com ênfase ainda na reconfiguração *on-the-fly* de instruções desta CPU. Este projeto faz parte de uma cooperação dentro de um projeto PROCAD/CAPES, entre o CIn-UFPE e LSC-UNICAMP.

1.1 OBJETIVOS

Este trabalho tem como objetivo o estudo de uma arquitetura dinamicamente reconfigurável para prototipação rápida de sistemas digitais baseada na CPU Leon[Gaisler Research]. Assim, a partir de uma plataforma pré-estabelecida, será estudada uma arquitetura que propõe a reconfiguração dinâmica parcial de instruções de uma configuração Leon[Gaisler Research], em uma plataforma Virtex-II[Menec Design].

Nesta proposta, um componente Virtex-II hospedará o core da CPU Leon e toda a sua estrutura de integração. Um core básico da CPU estará sempre presente (fixa) no sistema, enquanto que parte de sua estrutura de instruções (parte configurável) pode ser dinamicamente alterada, sem necessidade de *reset* do sistema (reconfiguração *on-the-fly*). Assim, novos códigos podem ser facilmente substituídos como se fossem cores, sem que haja necessidade uma reprogramação de todo um FPGA, reduzindo significativamente o tempo de prototipação e de testes de novos sistemas.

A escolha aqui do processador Leon se deu devido a sua portabilidade. Esta CPU Leon, uma arquitetura SPARC V8, tem seu código-fonte disponibilizado através da licença GNU LGPL, o que permite sua utilização gratuita e ilimitada para aplicações de pesquisa e comercial. Seu código fonte é descrito através de um modelo VHDL sintetizável, e totalmente configurável. Dentre as suas principais vantagens podemos citar a facilidade de

alterações, como também, a sua enorme adequação ao desenvolvimento de *SoCs*, dada a facilidade de adição de novos módulos.

A plataforma alvo para o desenvolvimento deste projeto é a plataforma DS-KIT-MBLAZE-V2 com uma FPGA VIRTEX-II, com 1.000.000 *gates* e periféricos[Menec Design], e a CPU Leon. O ambiente de desenvolvimento de software será o ISE6.1 para hardware.

1.2 ESTRUTURA DO TRABALHO

O trabalho se encontra divido em sete capítulos. O primeiro capítulo tem como objetivo mostrar a idéia geral do trabalho, os objetivos a serem atingidos e a estrutura de divisão dos capítulos. No segundo capítulo serão descritos os detalhes da CPU Leon, sua arquitetura geral e o funcionamento de cada um dos seus módulos. O capítulo 3 tem como objetivo fornecer uma base teórica para os dispositivos lógicos programáveis. Este capítulo descreve a evolução dos dispositivos lógicos, dando ênfase as FPGAs. A reconfigurabilidade dinâmica é tratada no capítulo 4. Neste capítulo também será abordado o processo de reconfiguração dinâmica das FPGAs da família Virtex-II. No capítulo 5 serão estudados os dispositivos de hardware e as ferramentas de software utilizadas. A arquitetura para reconfiguração estática e dinâmica de instruções é definida no capítulo 6, onde também é apresentado um estudo de caso para o modelo definido. E finalmente no último capítulo, apresentamos conclusões e trabalhos futuros.

2 O PROCESSADOR LEON 2

2.1 VISÃO GERAL

O processador Leon 2 é disponibilizado como um modelo VHDL sintetizável, totalmente configurável e que implementa um processador de 32-bit de acordo com a arquitetura SPARC V8. Seu código-fonte é disponibilizado através da licença GNU LGPL, que permite a utilização gratuita e ilimitada para aplicações de pesquisa e comercial. Seu modelo é bastante adequado ao desenvolvimento de System-on-Chip (*SOCs*), isto é, aplicações completas encapsuladas em um único chip. Além disso, possui a enorme vantagem de ser um processador de código-fonte aberto e totalmente configurável, possibilitando uma total liberdade de adaptação às aplicações e adição de novos módulos.

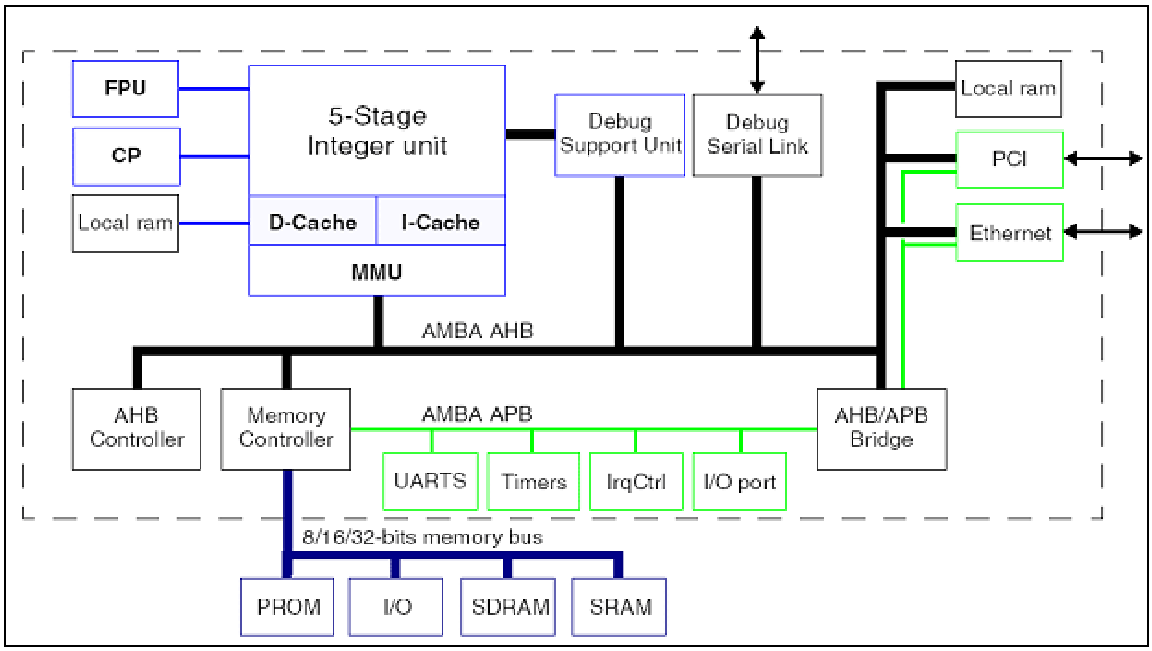


Figura 1 - Arquitetura do processador Leon

Como podemos ver no diagrama em blocos da *Figura 1*, esse processador é formado por uma unidade de inteiros com um pipeline de 5-estágios, por uma unidade de ponto-flutuante (*FPU – Float-Point Unit*) e uma interface para co-processador opcionais. A

unidade de inteiros implementa o padrão completo de instruções SPARC V8, inclusive as instruções de multiplicação e divisão. Em relação à unidade de ponto-flutuante, o modelo prover uma interface para a GRFPU (Gaisler Research Float Point Unit), para a Meiko FPU, disponibilizada pela Sun Microsystems, e para a incompleta LTH FPU, que não implementa todas as instruções SPARC V8. Além disso, o modelo também prover uma interface genérica para adição de co-processadores customizados.

Ainda em relação à arquitetura do Leon, temos duas caches multi-conjunto, uma de dados (*D-Cache*) e outra de instruções (*I-Cache*), uma unidade de gerenciamento de memória (*Memory Controler*), que quando habilitada, permite a utilização de um micro sistema operacional, uma unidade de depuração (*Debug suport unit*), que possibilita a utilização de ***breakpoints*** atém de acesso a todos os registradores internos na depuração dos programas, tudo isso sem afetar a performance do processador.

A interface flexível da memória, promove um mapeamento direto com a PROM, mapeamentos dos dispositivos de I/0, memória RAM estática e memória RAM dinâmica síncrona (SDRAM). Temos também disponível, dois temporizadores e um *watchdog*, ambos de 24 bits, duas UARTs de 8-bits, e um controlador de interrupção, responsável por gerenciar 15 interrupções internas e externas, e uma porta de I/0 paralela.

A arquitetura também implementa os barramentos AMBA, AHB e APB, que são responsáveis pela comunicação entre os dispositivos do sistema, tornado simples a adição de novos IP's cores. Opcionalmente podemos utilizar uma interface PCI, uma Ethernet 10/100 Mbit MAC e uma pequena RAM on-chip, configurável de 1-64 KByte.

2.2 ARQUITETURA

2.2.1 UNIDADE DE INTEIROS

A unidade de inteiros do Leon é caracterizada por um pipeline de 5-estágios, caches separadas de instruções e dados, número configurável de registradores entre 2 e 32, suporte as instruções de multiplicação e divisão e uma instrução opcional de MAC (16x16 bit e acumulador de 40 bits).

A *Figura 2*, abaixo, mostra o diagrama em bloco da unidade de inteiros do Leon.

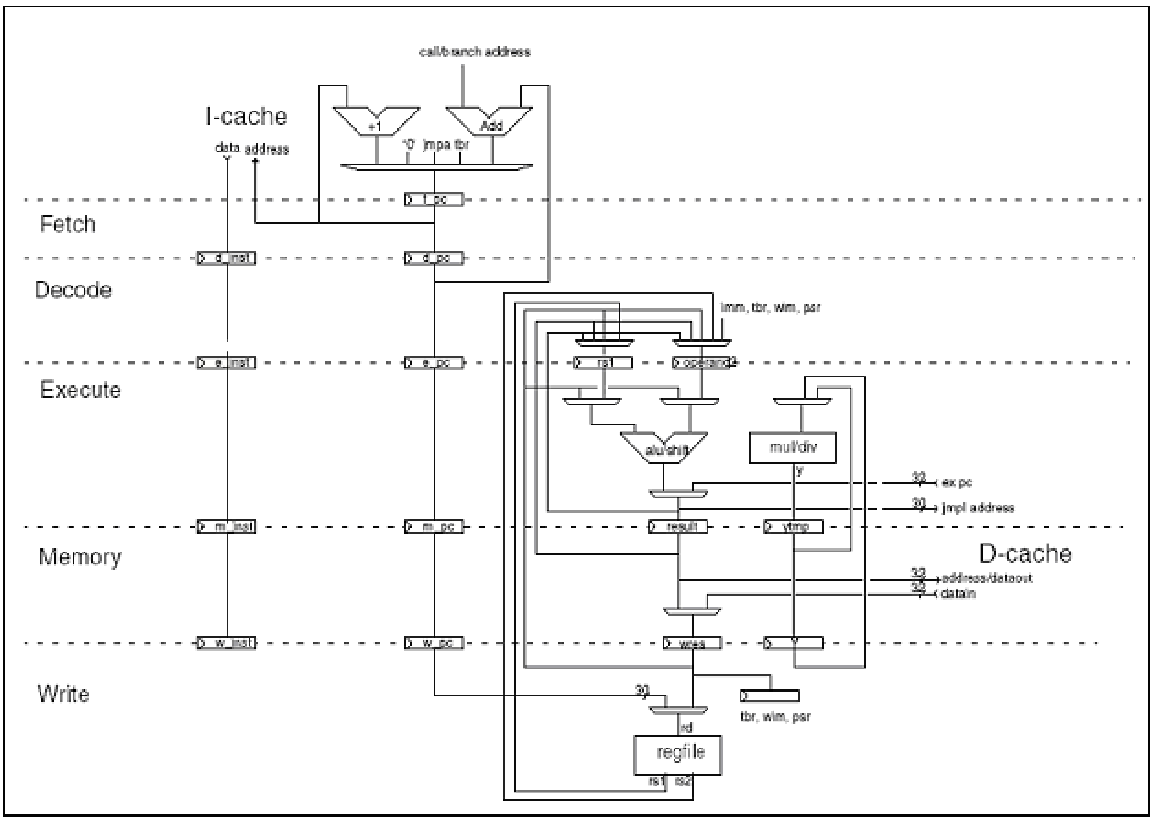


Figura 2 - Diagrama em blocos da Unidade de Inteiros do Leon

Como podemos ver na figura acima, os 5 estágios do pipeline da unidade de inteiros do Leon, são: Busca de Instrução, Decodificação, Execução, Memória e Escrita. No estágio de Busca de Instrução (FE - Intruction Fetch), obtemos a instrução a ser decodificada da cache de instruções, caso esta esteja habilitada, caso contrário, a busca será encaminhada ao controlador de memória.

O estágio Decodificação (DEC - Decode) é responsável por decodificar a instrução, assim como a leitura dos operandos. Os endereços alvos das instruções de CALL e Branch também são gerados nesse estágio.

No estágio de Execução (EX - Execute), operações da ALU, operações lógicas e de Shift são executadas. Para as operações de memória e JMPL/RETT, os endereços são gerados.

No estágio de Memória (ME - Memory) a cache de dados é acessada, e os dados estarão válidos ao final deste estágio. Dados armazenados, lidos no estágio de execução são escritos na cache de dados neste momento.

Por último, no estágio de Escrita (WR - Write) o resultado de qualquer operação da ALU, operações lógicas, de deslocamento, ou operações de leitura de cache, são escritas de volta no banco de registradores.

Agora que já descrevemos o funcionamento geral do pipeline da unidade de inteiros do Leon, vamos voltar nossa atenção, para as instruções de multiplicação, as de multiplicação e acumulação (MAC – Multiply&Acumulate) e as instruções de divisão, suportadas por esse modelo.

Começamos nossa discussão com as instruções de multiplicação de inteiros. Há um suporte completo as instruções UMUL, SMUL, UMULCC e SMULCC, definidas no padrão SPARC. Estas instruções recebem operandos de 32-bit e produzem como resultado um valor de 64-bit. UMUL e UMULCC fornecem multiplicação sem sinal enquanto SMUL e SMULCC, multiplicação sinalizada. UMULCC e SMULCC utilizam códigos de condição a fim de refletir o resultado.

Em relação à multiplicação e acumulação, temos duas instruções implementadas: UMAC e SMAC, ambas utilizam um acumulador de 40-bit. Estas instruções são executadas em um único ciclo, mas possuem dois ciclos de latência.

Por fim, temos também implementadas as seguintes instruções de divisão: SDIV, UDIV, SDIVCC, UDIVCC. Estas instruções, assim como, questões de arredondamentos e detecção de overflow, também estão definidas segundo o padrão SPARC V8.

A arquitetura SPAPC V8 define dois co-processadores opcionais: uma unidade de ponto flutuante e um co-processador customizado. O pipeline do Leon fornece uma interface para cada uma destas unidades. Podemos conectar três unidades de ponto flutuante diferentes, são elas: Gaisler Research's GRFPU, Meiko FPU da Sun MicroSystems e a LTH FPU.

A GRFPU e a Meiko FPU operam com precisão simples e dupla e implementam todas as instruções de ponto-flutuante definidas no padrão SPARC V8. Isto não ocorre com a LTH FPU, esta, por não implementar todas instruções, não pode ser utilizada em programas de propósito geral, sendo considerada, por isso, incompleta.

As instruções de ponto flutuante não são executadas em paralelo com as instruções da unidade de inteiros na Meiko FPU e na LTH FPU. Assim, o processador é parado durante a execução destas instruções, diferentemente da GRFPU, que é completamente paralela, e permite o início de uma nova instrução a cada ciclo de clock. As únicas exceções ocorrem com as instruções FDIV e FSQRT, que só podem ser executadas uma de cada vez, embora sejam executadas em uma unidade de divisão separada, não bloqueando a FPU de processar todas as outras operações em paralelo. Ainda faz parte da unidade de inteiros, uma operação de Reset, responsável por reiniciar o processador, de 1 a 4 ***breakpoints*** de hardware e tratamento de exceções. Informações adicionais sobre a unidade de inteiros do Leon pode ser encontrada em [Gaisler Research].

2.2.2 SUBSISTEMA DE CACHE

O processador Leon implementa uma arquitetura HARVAD, com barramentos de instruções e dados separados, conectados a dois controladores de cache independentes. Estes controladores de cache podem ser configurados separadamente, podendo implementar caches de mapeamento direto ou multi-conjunto com associatividade entre 2 e 4 conjuntos. Quanto ao tamanho, podemos configurá-las de 1-64Kbyte, divididas em linhas de cache de 8-32 bytes de dados. Na configuração multi-conjunto, uma das três políticas de substituição de cache podem ser utilizadas: LRU, LRR ou pseudo-randômico. Se o algoritmo LRR for utilizado, a cache terá que ser associativa de duas vias.

As operações das caches de instruções e dados são controladas através de um Registrador de Controle de Cache comum (CCR). Cada uma das caches pode estar em um dos três modos: desabilitada, habilitada e congelada. Se esta estiver desabilitada, nenhuma operação de cache é fornecida e as operações de ***load*** e ***store*** são passadas diretamente para o controlador de memória. Caso esteja congelada, esta será acessada e colocada em sincronismo com a memória principal, como se ela estivesse habilitada, mas nenhuma nova linha será alocada em leituras perdidas.

2.2.3 UNIDADE DE GERENCIAMENTO DE MEMÓRIA

O modelo do processador Leon permite a utilização opcional de uma unidade de gerenciamento de memória (MMU) compatível com o padrão SPARC V8. Caso esta unidade de gerenciamento de memória esteja desabilitada, as memórias caches irão operar normalmente utilizando mapeamento físico de endereços. Caso contrário, os rótulos das caches armazenaram o endereço virtual, assim como, um campo de contexto de 8-bit. Detalhes de operação da unidade de gerenciamento de memória são encontrados em [SPARC]. A MMU também pode ser configurada para utilizar uma TLB compartilhada, ou uma TLB para instruções e outra pra dados. O número de entradas da TLB pode ser configurado entre 2-32.

2.2.4 BARRAMENTOS

Dois barramentos internos são fornecidos: AMBA AHB e APB. O APB é utilizado para acessar registradores internos e funções periféricas, enquanto o AHB é utilizado para transferência de dados em alta velocidade . A especificação completa pode ser encontrada em [ARM Limited].

O Leon utiliza o barramento AMBA-2.0 AHB para conectar os controladores de cache do processador ao controlador de memória e outras unidades de alta velocidade. Na configuração padrão, o processador é o único mestre do barramento, enquanto o controlador de memória e a ponte APB são escravos. A Tabela 1 abaixo, mostra a alocação padrão de endereços AHB.

Address range	Size	Mapping	Module
0x00000000 - 0x1FFFFFFF 0x20000000 - 0x3FFFFFFF 0x40000000 - 0x7FFFFFFF	512 M 512 M 1 G	Prom Memory bus I/O SRAM and/or SDRAM	Memory controller
0x80000000 - 0x8FFFFFFF	256 M	On-chip registers	APB bridge
0x90000000 - 0x9FFFFFFF	256 M	Debug support unit	DSU
0xA0000000 - 0xA001FFFF	128 K	PCI control registers	PCI
0xB0000000 - 0xB001FFFF	128 K	Ethernet MAC registers	Ethernet
0xC0000000 - 0xFFFFFFFF	1G	PCI bus	PCI

Tabela 1- Alocação padrão de endereços AHB

Já a ponte APB é conectada ao barramento AHB como escravo mas age como único mestre no barramento APB. A maioria dos periféricos internos são acessados através do barramento APB. O processador é conectado ao barramento AHB através dos controladores de cache de dados e de instruções, como mostra a arquitetura da figura 1.

Informações adicionais sobre a CPU Leon podem se encontradas em [Gaisler Research].

3 DISPOSITIVOS LÓGICOS PROGRAMÁVEIS

3.1 INTRODUÇÃO

O processo de desenvolvimento de hardware digital sofreu grandes modificações nos últimos anos. Diferentemente das tecnologias anteriores, onde o desenvolvimento era feito utilizando *chips SSI* contendo algumas portas lógicas básicas, quase a maior parte do desenvolvimento digital atual consiste na utilização de dispositivos de alta densidade. Devido ao alto custo e longo tempo de fabricação dos *gate arrays NRE (*nonrecurring engineering*)*, a maioria dos protótipos, como também produção de sistemas digitais, vêem utilizando dispositivos lógicos programáveis em campo. As principais vantagens de se utilizar estes dispositivos são o baixo custo de desenvolvimento, alta velocidade de produção, baixo risco financeiro, velocidade de operação compatível com a velocidade do produto final, assim como maior facilidade de modificação.

3.2 EVOLUÇÃO DOS DISPOSITIVOS LÓGICOS

O primeiro tipo de dispositivo, programável pelo usuário, que poderia implementar circuitos lógicos, foram as *PROM* (Programmable Read-Only Memory). Por se mostrar uma arquitetura ineficiente na implementação de circuitos lógicos, raramente são utilizadas para esse propósito [Brown; Rose]. Mais tarde, surgiram os *FPLA* (Field-Programmable Logic Array) ou *PLAs*, que foram desenvolvidos especificamente para a implementação de circuitos lógicos. Estes dispositivos consistem em dois níveis de portas lógicas, um array de ANDs programável, seguido por um array de OR programável. Um PLA é estruturado de forma que qualquer de suas entradas, ou seu complemento, possam ser associados ao array de ANDs. Desta forma, cada saída do array de ANDs, pode corresponder a qualquer produto de termos da entrada. De forma similar, cada saída do array de ORs, pode ser configurada, a fim de produzir uma soma lógica das saídas do array de ANDs. Através desta estrutura, um PLA, será capaz de implementar funções lógicas da foram de soma de produtos. Quando estes dispositivos foram introduzidos, no início dos anos 70, pela Philips, os principais problemas eram o custo de fabricação, e baixa performance. Para suprir estas

deficiências surgiram os *PAL* (Programmable Array Logic). Estes dispositivos possuem um array de ANDs programável e um array de ORs fixo. de A figura abaixo ilustra estas características.

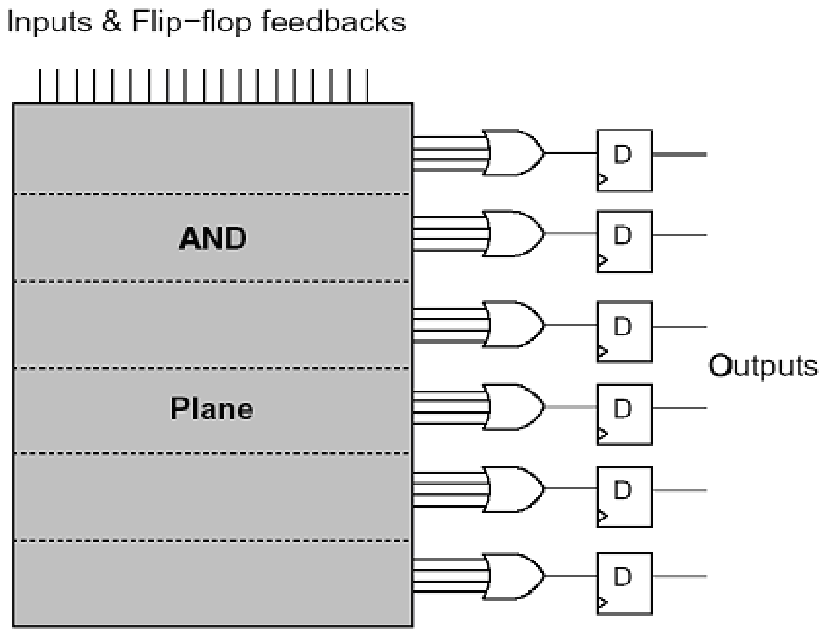


Figura 3 – Estrutura de um PAL

Todos os pequenos *PLDs*, incluindo *PLAs*, *PALs* e dispositivos semelhantes estão agrupados em uma única categoria, chamada *Simple PLDs* (SPLDs), onde as principais características destes dispositivos são: baixo custo e alta performance pino-a-pino.

À medida que tecnologia avançou, tornou-se possível à produção de dispositivos com mais capacidade que os *SLPDs*. Então, através de integração de múltiplos *SPLDs* em um único chip, surgiram os *Complex PLDs* (CPLDs). Estes dispositivos fornecem capacidade lógica equivalente a mais de 50 dispositivos *SPLDs* típicos. Contudo, a extensão desta arquitetura a densidades mais altas se mostrou bastante difícil. Assim, a fim de construir dispositivos com capacidades lógicas elevadas, uma nova metodologia era necessária.

Os dispositivos de propósito gerais, disponíveis atualmente, de maior capacidade lógica são os tradicionais *Gate Arrays*, algumas vezes chamados de *MPGA* (Mask – Programmable Gate Array). Estes dispositivos consistem em um array de transistores pré-

fabricados que podem ser customizados para o circuito lógico do usuário, através da conexão dos transistores. Esta customização é feita durante a fabricação do chip, através especificação do metal utilizado na interconexão. Isto significa um alto custo de configuração e um longo tempo de fabricação. Embora *MPGAs* não sejam *FPDs*, a razão pelo qual o mencionamos aqui, é devido ao fato destes dispositivos terem motivado o desenvolvimento de outros dispositivos equivalentes, as FPGAs (Field Programmable Gate Arrays). Assim como as *MPGAs*, as FPGAs são formadas por blocos lógicos e interconexões sobre estes blocos, a diferença está na configuração, através de programação pelo usuário final. A figura a seguir mostra uma arquitetura típica de uma FPGA.

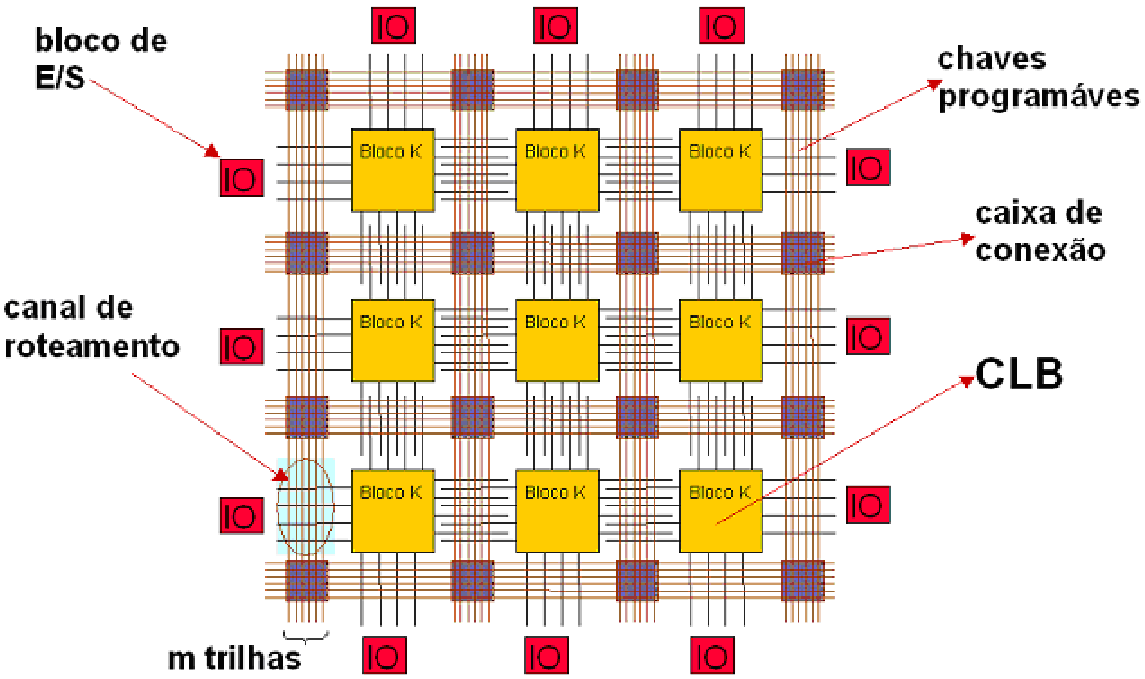


Figura 4 – Estrutura de uma FPGA

Por ser o único tipo de *FPD* que suporta capacidades lógicas elevadas, os FPGAs são responsáveis pela maior parte das mudanças ocorridas na maneira como os circuitos digitais são desenvolvidos atualmente [Brown; Rose].

A seguir entraremos em mais detalhes sobre estes dispositivos, já que o nosso estudo de caso utilizara um destes dispositivos.

3.3 FPGAS DA FAMÍLIA VIRTEX-II

Os dispositivos da família Virtex II são *gate arrays* programáveis pelos usuários, com vários elementos configuráveis. Sua arquitetura é otimizada para altas densidades e desenvolvimento lógico de alta performance. Os dispositivos programáveis abrangem: blocos de entrada e saída, os *IOBs* e blocos lógicos configuráveis, os *CLBs*. A figura abaixo mostra a arquitetura geral destes dispositivos.

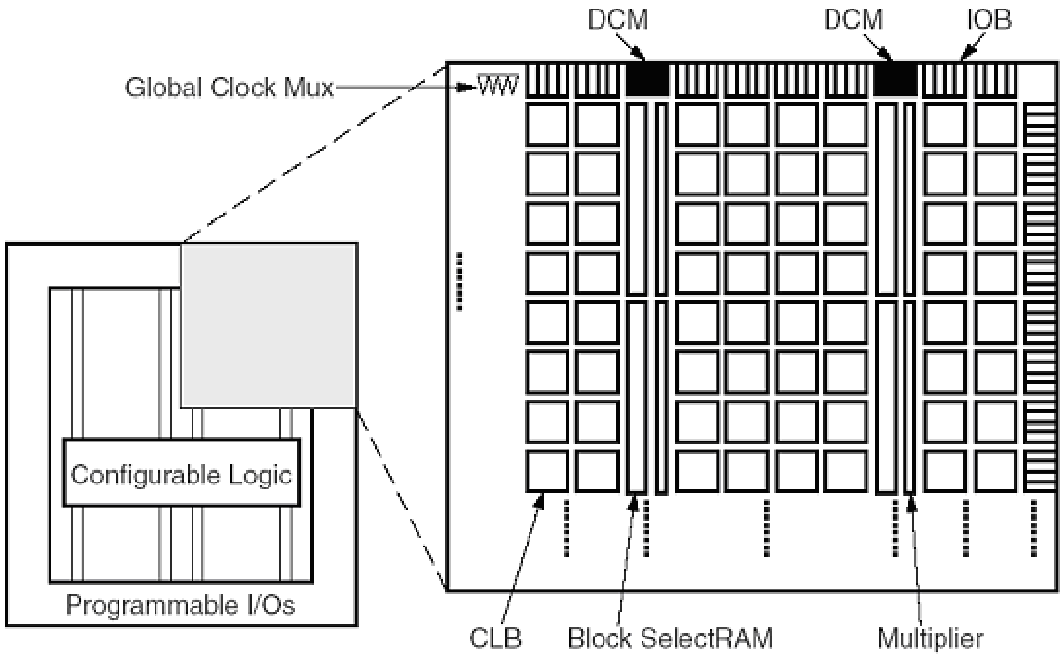


Figura 5 - Visão geral da arquitetura Virtex-II

Na arquitetura descrita acima, podemos visualizar os *IOBs*, responsáveis por fornecer uma interface de comunicação entre pinos e a lógica interna configurável. Estes blocos de entrada e saída, suportam desde padrões de entrada mais populares a padrões de tecnologia de ponta, desta forma, podem ser categorizados em: blocos de entrada com registrador de taxa de dados simples ou registrador de taxa de dados dupla (DDR) , blocos de saída com registrador opcional de taxa de dados simples ou taxa de dados dupla, e um *buffer 3-state* opcional; e como bloco bidirecional, ou seja, qualquer combinação de entradas e saídas.

O outro elemento configurável da arquitetura Virtex-II, são os chamados Blocos Lógicos Configuráveis (CLBs). Estes elementos são formados por quatro *slices* equivalentes e dois *buffers 3-state*. Cada *slice* contém dois geradores de funções (F&G), dois elementos de armazenamento, portas lógicas aritméticas, multiplexadores, permitindo uma vasta capacidade de funções,. Os geradores de funções são configurados como *Look-Up-Tables* (LUTs) de quatro entradas, como registradores de deslocamento de 16-bit ou como memória SelectRAM distribuída de 16-bit. Alem disso, os dois dispositivos de armazenamento são *flip-flops* tipo D trigado na mudança ou *Letchs* sensível a nível. Cada *CLB* contém interconexões internas rápidas, e conexões para a matriz de comutação (switch matrix), para acessar recursos gerais de roteamento. Na tabela 2 abaixo podemos encontrar informações a respeito das FPGA da família Virtex-II. O kit de desenvolvimento alvo, utiliza uma FPGA de 1M *gates* equivalentes, que de acordo com a tabela abaixo, possui 5120 s*lices* e um número máximo de 432 I/O *pads*. Características de outros dispositivos da família Virtex-II, podem ser visto abaixo.

Device	System Gates	CLB (1 CLB = 4 slices = Max 128 bits)			Multiplier Blocks	SelectRAM Blocks		DCMs	Max I/O Pads[1]
		Array Row x Col.	Slices	Maximum Distributed RAM Kbits		18 Kbit Blocks	Max RAM (Kbits)		
XC2V40	40K	8 x 8	256	8	4	4	72	4	88
XC2V80	80K	16 x 8	512	16	8	8	144	4	120
XC2V250	250K	24 x 16	1,536	48	24	24	432	8	200
XC2V500	500K	32 x 24	3,072	96	32	32	576	8	264
XC2V1000	1M	40 x 32	5,120	160	40	40	720	8	432
XC2V1500	1.5M	48 x 40	7,680	240	48	48	864	8	528
XC2V2000	2M	56 x 48	10,752	336	56	56	1,008	8	624
XC2V3000	3M	64 x 56	14,336	448	96	96	1,728	12	720
XC2V4000	4M	80 x 72	23,040	720	120	120	2,160	12	912
XC2V6000	6M	96 x 88	33,792	1,056	144	144	2,592	12	1,104
XC2V8000	8M	112 x 104	46,592	1,456	168	168	3,024	12	1,108

Tabela 2 – Família Virtex-II

4 RECONFIGURABILIDADE DINÂMICA

Este capítulo aborda a reconfigurabilidade dinâmica de hardware em dispositivos lógicos programáveis. Na primeira seção será apresentada uma introdução sobre o tema, assim como alguns conceitos relacionados. Em seguida será feito um estudo sobre os tipos de reconfiguração dinâmica existentes. Na seção 3, será discutido o processo de reconfiguração dinâmica nas FPGAs da família Vitex-II, e na última parte do capítulo, algumas limitações e problemas.

4.1 INTRODUÇÃO A RECONFIGURABILIDADE DINÂMICA

Desenvolvimentos recentes na tecnologia digital reconfigurável têm introduzido suporte a alterações parciais do hardware, sem que o funcionamento do restante da lógica seja prejudicado. Ou seja, o hardware pode ser alterado dinamicamente sem a necessidade de ser desligado.

Sistemas reconfiguráveis são plataformas onde a arquitetura pode ser modificada em tempo real, para melhor se adequar à aplicação que será executada, permitindo uma maior eficiência na execução de aplicações específicas, do que as conseguidas através de processadores de uso geral [Ribeiro].

Fazendo-se uma analogia com o conceito de memória virtual, onde os programas armazenados e suas estruturas de dados deixam de estar limitados ao tamanho da memória física disponível. De forma semelhante, a capacidade lógica de um dispositivo reconfigurável pode ser virtualmente aumentada, permitindo que múltiplos conjuntos de configurações sejam armazenados em memórias e carregados à medida que sejam necessários, possibilitando uma utilização mais eficiente do espaço físico disponível no dispositivo [Mesquita]. Este conceito é freqüentemente chamado de *hardware virtual*, e inúmeras aplicações podem ser tornar possível através deste recurso, que alem de possibilitar o aumento da capacidade lógica, também pode ser utilizado na aceleração de áreas específicas de um programa, através do chaveamento de hardware, aumentando

assim, a eficiência da certas operações executadas. Por fim, também possibilita à redução de consumo de potência. A Figura 5 abaixo, ilustra este conceito de *hardware virtual.*

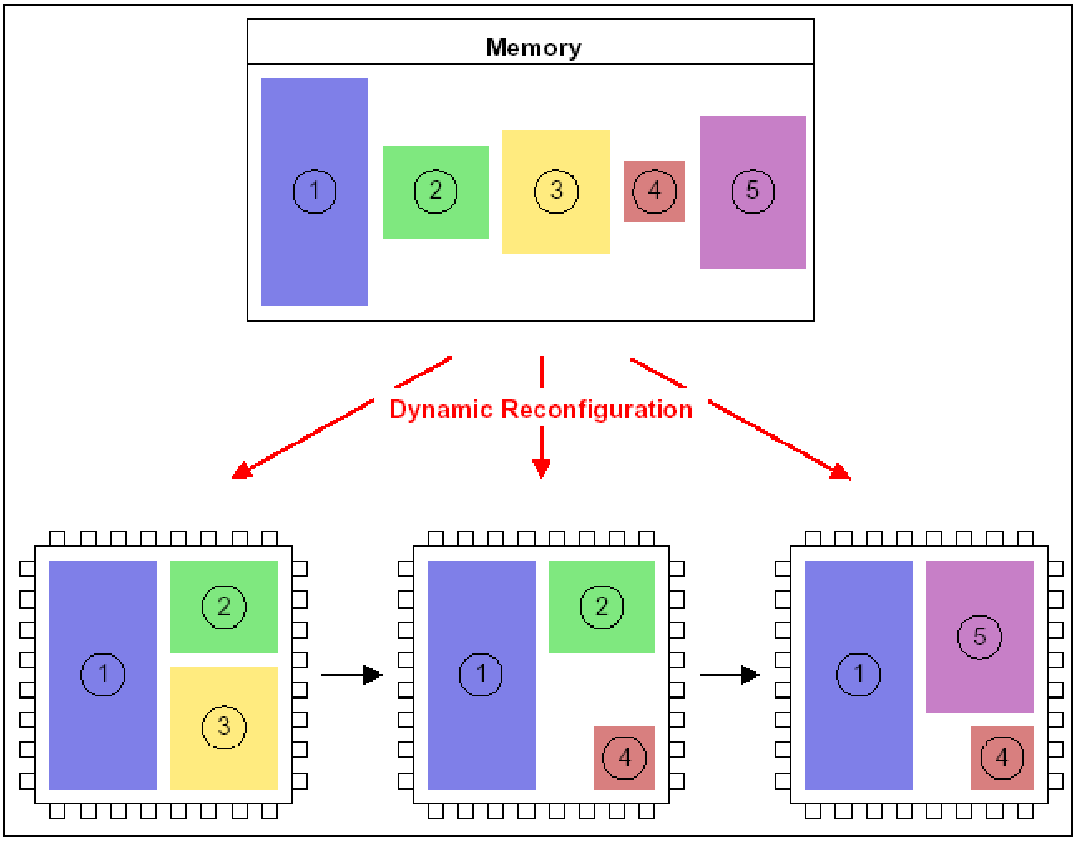


Figura 6 – Exemplo de hardware Virtual

A reconfigurabilidade dinâmica do hardware torna possível inúmeras aplicações em diversas áreas, como na medicina, nas telecomunicações, em redes de computadores e muitas outras áreas, onde a disponibilidade é um requisito altamente importante. Através deste recurso, estes dispositivos poderão ser atualizados dinamicamente, sem a necessidade de interromper o funcionamento do hardware.

4.2 TIPOS DE RECONFIGURAÇÃO DINÂMICA

O controle de reconfiguração é um dos pontos fundamentais de uma arquitetura reconfigurável [Boschetti]. Este tema pode ser tratado de duas maneiras: reconfiguração dinâmica parcial e reconfiguração dinâmica multicontexto.

A reconfiguração dinâmica parcial se refere à redefinição de partes de um circuito mapeado em um dispositivo lógico programável através de mudanças no ***bitstream*** de configuração. Neste tipo de reconfiguração, a estrutura interna do dispositivo programável, no nosso caso os FPGAs, é dividida em uma série de colunas, sendo possível alterar a configuração de uma ou várias colunas através de modificações no arquivo de configuração. Uma desvantagem desta abordagem é que à medida que a quantidade de alterações no circuito se tornam grandes, o tamanho do ***bitstream*** pode crescer muito e causar perda de desempenho devido ao tempo de reconfiguração. As FPGAs da família Virtex utilizam este tipo de reconfiguração.

Já a reconfiguração dinâmica multicontexto refere-se ao armazenamento de grupos de configuração na FPGA [Boschetti], chamados de planos de configuração. Nesta abordagem, um plano que se encontra ativo em determinado momento pode ser chaveado por outro plano rapidamente, de acordo com a aplicação. A desvantagem deste modelo é a grande quantidade de área reservada para os contextos, podendo limitar significativamente a área restante para a lógica ativa. Porém, devido a grande capacidade lógica dos dispositivos atuais, esta desvantagem pode não se considerada tão critica.

4.3 RECONFIGURAÇÃO DINÂMICA EM FPGAS VIRTEX-II

As FPGAs Vitex-II utilizam o tipo de reconfiguração parcial, como mencionado na seção anterior. Há vários modos possíveis pra esse tipo de reconfiguração, dentre eles podemos citar os modos *Slave SelectMAP* e *Boundary Scan* (JTAG). Através destes tipos de reconfiguração, o novo dado é carregado em uma área reconfigurável específica do dispositivo, enquanto a lógica localizada na parte não configurável ou parte fixa, continua operando. Nestas FPGAs os dados são carregados baseados em colunas, onde a menor unidade de carga é um ***bitstream*** de configuração, chamado de ***frame***. O tamanho do ***frame*** varia de acordo como o dispositivo utilizado. O dispositivo XC2V8000 da família Virtex-II

contém 9.152 bits de dados de configuração por *frame* e possui sua memória de configuração dividida em 2860 *frames* [Ribeiro, Marques].

As FPGAs Virtex podem utilizar um dos três tipos de reconfiguração parcial descritos a seguir, são eles: reconfiguração parcial multi-coluna com módulos independentes entre si, reconfiguração parcial multi-coluna com comunicação entre os módulos e um outro tipo de reconfiguração, chamado de *Small Bit Manipulation*. A reconfiguração parcial multi-coluna com módulos totalmente independentes é recomendado a aplicações onde não há comunicação entre os módulos do sistema, e desta forma a reconfiguração de um módulo não afeta a operação dos outros módulos. Na reconfiguração parcial multi-coluna com comunicação entre os módulos, há um barramento especial responsável pelo roteamento entre os módulos. Este barramento especial é chamado de *Bus Macro* e será discutido em seguida. O último tipo de reconfiguração, a Small Bit Manipulation, é caracterizado por gerar *bitstream* baseado apenas nas diferenças entre os módulos configuráveis. Desta forma, a troca entre as configurações é feita de maneira bastante rápida, pelo fato de que as diferenças entre os *bitstreams* de configuração serem bem menores que o do sistema completo.

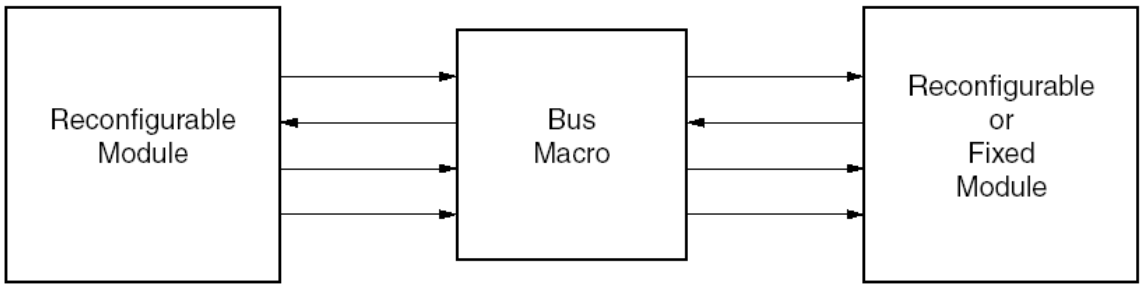


Figura 7 – Bus Macro utilizada para comunicação inter-módulo

A *Figura 6* acima, mostra dois módulos que podem ser reconfiguráveis e um módulo intermediário funcionando como "ponte" de roteamento fixo entre os dois módulos. Este módulo intermediário é chamado de *Bus Macro* e se caracteriza por ser pré-roteda e por sua utilização na especificação de canais de roteamento exato. Isto significa que este módulo não sofre modificações durante as mudanças das configurações. Uma implementação física da *Bus Macro* pode ser vista na Figura 7, a seguir.

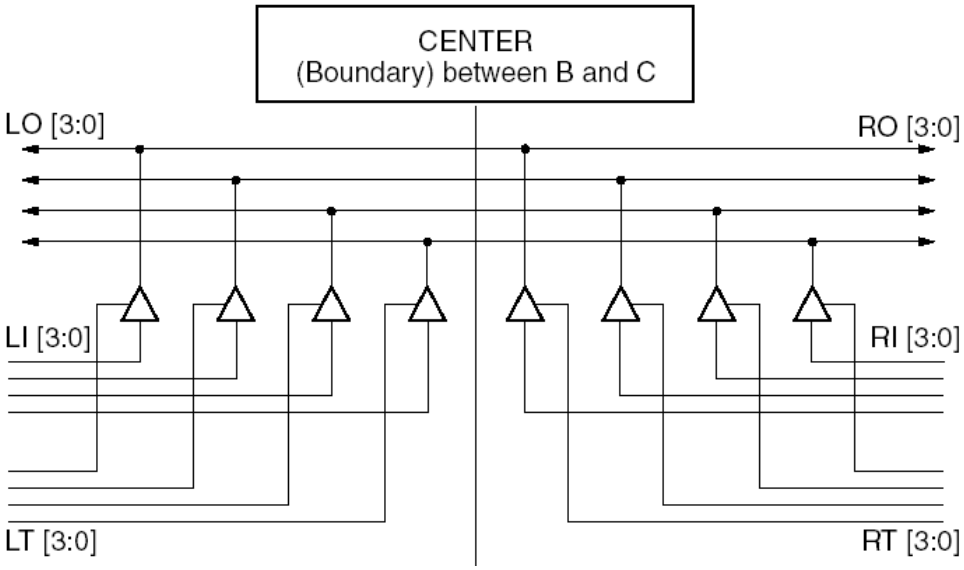


Figura 8 – Implementação Física da *Bus Macro*

4.4 LIMITAÇÕES E PROBLEMAS

Entretanto há algumas limitações implícitas a arquitetura das FPGAs que podem inviabilizar a reconfiguração dinâmica, ou pelo menos limitar as suas aplicações. A primeira destas limitações é o tempo de reconfiguração do dispositivo, ou seja, o tempo para carregar uma configuração na FPGA pode limitar a sua aplicabilidade em relação aos processadores de propósito geral. Um outro problema é a restrição de tamanho que algumas ferramentas CAD, disponíveis no mercado impõem, limitando o tamanho dos *cores* aos recursos disponíveis no dispositivo. Podemos também citar a complexidade do desenvolvimento, assim como a falta de suporte eficiente fornecida pelos fabricantes. Com isso, podemos concluir que as ferramentas CAD disponíveis no mercado, ainda não estão totalmente preparadas para possibilitar um bom fluxo de projeto reconfigurável.

5 PLATAFORMA DE DESENVOLVIMENTO

Este capítulo descreve a plataforma alvo, os dispositivos de hardware disponíveis e as ferramentas de software envolvidas.

5.1 XILINX VIRTEX-II V2MB1000

O kit de desenvolvimento V2MB1000 é uma solução completa para desenvolvimento de aplicações baseadas na família de FPGAs Virtex-II da Xilinx. Este kit utiliza uma *FPGA* de 1 milhão de *gates* (XC2V1000-4FG456C), uma memória externa DDR 16Mx16, duas fontes de *clock*, uma porta RS-232, além suporte a circuitos adicionais. A figura abaixo apresenta o diagrama em blocos da arquitetura da placa.

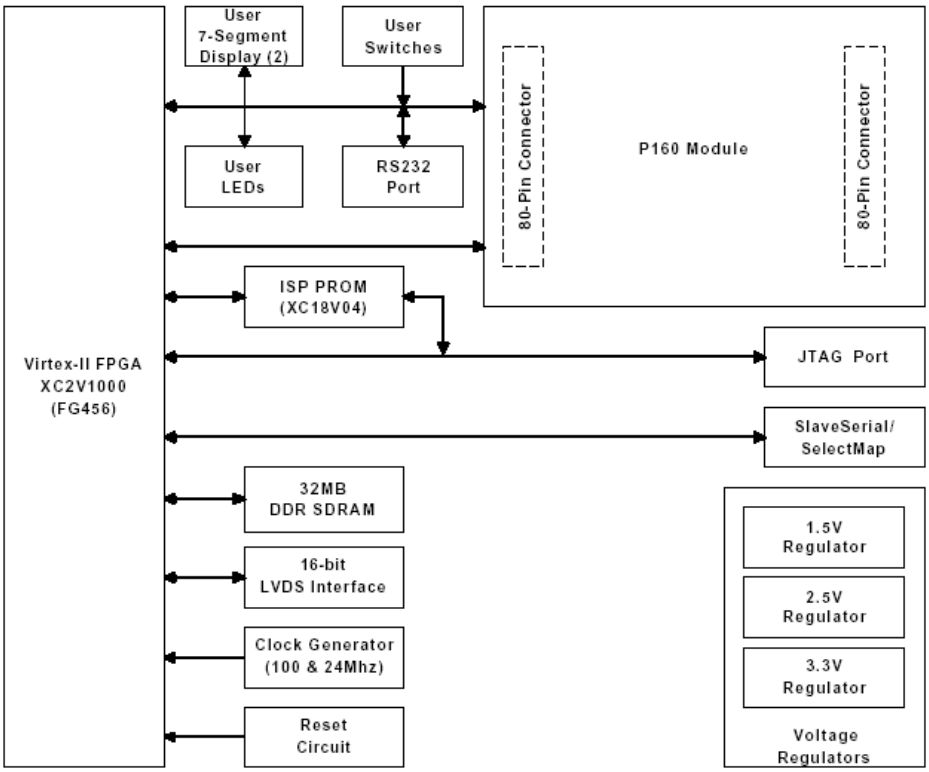


Figura 9 – Diagrama em blocos da placa de prototipação

A memória DDR de 32MB, disponível na placa, é implementada utilizando o dispositivo *Micro MT46v 16M16TG-75* e uma visão em alto nível desta interface DDR pode ser vista abaixo.

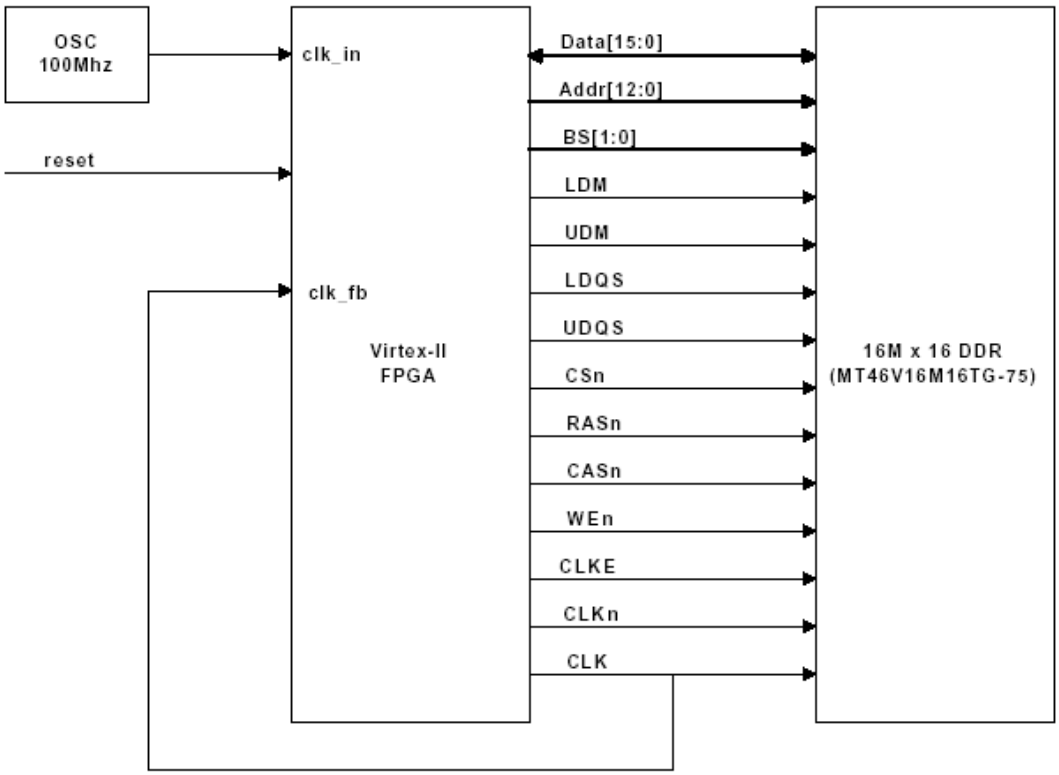


Figura 10 – Interface DDR

Além disso, também temos disponível dos osciladores internos, um de 100MHz e outro de 24MHz, habilitados através de *jumpers* disponíveis na placa. Temos também um circuito de *reset*, que supervisiona a tensão no dispositivo. Este circuito é responsável por *resetar* o dispositivo caso a tesão caia abaixo do mínimo especificado, ou seja, abaixo de 1,425 Volts. Outros recursos disponíveis, são os dois displays de 7segmentos, que podem ser utilizados em testes e depurações. A figura abaixo mostra a interface deste dispositivo.

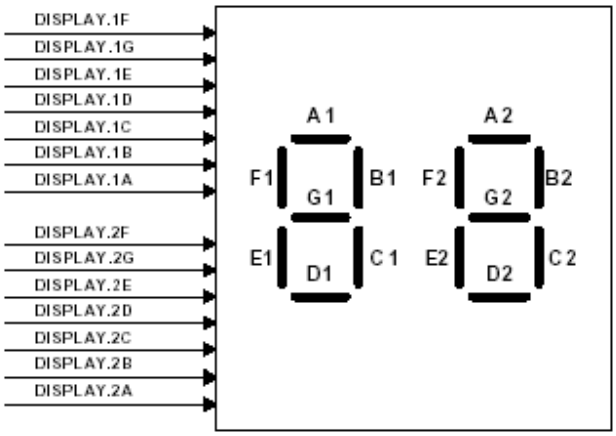


Figura 11 – Interface do Display de 7-Segmentos

Complementando os recursos do kit de desenvolvimento, há também um *LED*, dois *Push Buttons*, 8 chaves de entrada configuradas estaticamente em nível lógico alto ou baixo, uma porta RS-232 e um conector *JTAG* utilizado para programar a *ISP PROM* e para configurar a FPGA.

Informações adicionais sobre este kit de desenvolvimento podem ser encontradas em [Menec Design].

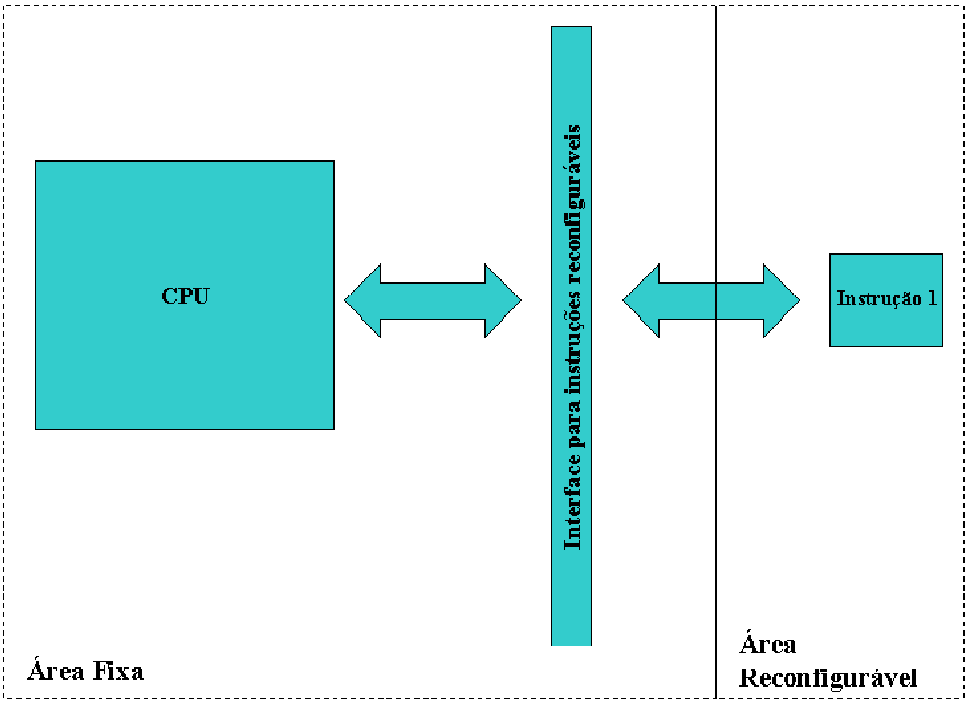
5.2 FERRAMENTAS DE SOFTWARE

Durante o estudo da CPU Leon, no capitulo 2 deste trabalho, relatamos algumas de suas vantagens e dentre elas citamos a sua capacidade de configuração. Como já foi dito, a CPU Leon é altamente configurável, permitindo a customização para certas aplicações e tecnologias alvos. Esta configuração pode ser realizada no próprio código-fonte, descrito em *VHD*, através de modificações diretas nos arquivos *"target.vhd"* e *"device.vhd"*, ou através da utilização de uma ferramenta gráfica fornecida juntamente com sua distribuição. Esta ferramenta gráfica de configuração está disponível para a plataforma Unix e para plataforma Windows/Cygwin, sendo bastante utilizada no processo de configuração deste processador. Através desta ferramenta pode-se configurar as memórias caches e suas políticas de substituição, a quantidade de registradores, assim como, habilitar e desabilitar módulos. No *Apêndice C* há uma descrição detalhada da utilização desta ferramenta dirigida à configuração proposta para esta CPU.

Após o processo de configuração da CPU, há a etapa de síntese, posicionamento e o roteamento. A ferramenta utilizada neste processo foi o ISE 6.1 da Xilinx. Uma descrição detalhada da utilização desta ferramenta pode ser encontrada em [XILINX].

6 DEFINIÇÃO DA ARQUITETURA

Este capítulo descreve a arquitetura proposta para a reconfiguração estática e dinâmica de instruções na CPU Leon. Esta arquitetura é formada por dois módulos alocados em uma área fixa da FPGA e um módulo localizado em uma região reconfigurável. Os módulos alocados nesta região fixa são a CPU Leon e o core da interface para instruções reconfiguráveis, discutido nas seções seguintes. Já o módulo de instrução, responsável pela funcionalidade da instrução, é alocado em uma área dinamicamente reconfigurável do dispositivo. A figura abaixo mostra uma visão em alto nível desta arquitetura. O objetivo é modificar a área reconfigurável, enquanto a área fixa continua a operar. Na figura abaixo o *core* da *instrução 1* é substituído pelo *core* da *instrução 2* enquanto a CPU se encontra em funcionamento.



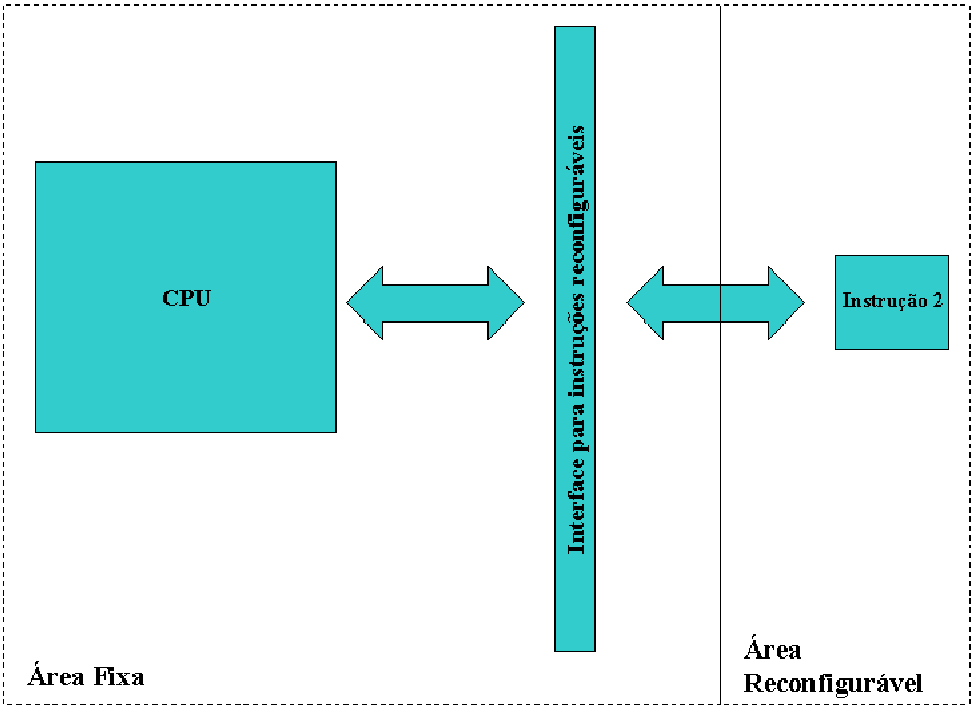


Figura 12 – Modelo da arquitetura proposta

Nas seções seguintes será detalhado o funcionamento de cada um dos módulos da arquitetura, e será realizado um estudo de caso para a plataforma especificada.

6.1 MÓDULO DA CPU

A CPU escolhida para a arquitetura proposta foi o Leon, principalmente por ser tratar de uma arquitetura RISC de código-fonte aberto, além de inúmeras outras vantagens, citadas no capitulo 2 deste trabalho. Para que seja possível implementar a reconfiguração dinâmica de instruções, há necessidade de algumas alterações nesta CPU. A primeira coisa a se fazer, é definir o tipo de instrução reconfigurável que será suportada pela arquitetura. Restringiu-se às instruções do tipo R, ou seja, operações lógico-aritméticas em registradores, definidas na arquitetura SPARC, referenciada em [SPARC]. Através destas instruções e da interface definida, não é possível escrever e ler da memória, sendo apenas permitido operações com os registradores. Feito isso, também foram definidos os sinais responsáveis pela comunicação entre este módulo e o módulo da interface para novas instruções.

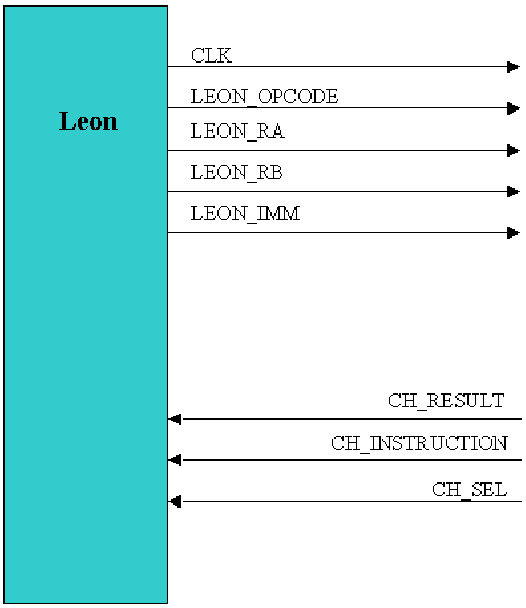


Figura 13 – Módulo da CPU

Na figura acima, podemos identificar as informações que são enviadas para a interface, assim como as recebidas desta. A CPU envia para o módulo da interface, o sinal de *clock* (*CLK*), responsável por sincronizar sua operação com os outros módulos, e os campos da instrução que está sendo executada, ou seja, o *opcode* (LEON_OPCODE), responsável por identificar a instrução, e outros campos definidos para o formato de instruções lógicas e aritméticas (LEON_RA, LEON_RB,LEON_IMM). Os sinais, *CH_RESULT, CH_INSTRUCTION e CH_SEL*, enviados pela interface para a CPU serão explicados na próxima seção, quando abordaremos o processo de reconfiguração.

6.2 MÓDULO DA INTERFACE PARA INSTRUÇÕES RECONFIGURÁVEIS

Este é o principal módulo da arquitetura proposta. Este módulo tem o objetivo fornecer uma interface padrão onde novas instruções possam ser adicionadas dinamicamente ao conjunto de instruções da CPU Leon. Esta interface pode funcionar em dois modos de operação: modo de execução e modo de reconfiguração. No modo de execução os endereços das novas instruções são decodificados e os sinais necessários a sua

execução são ativados. Caso não exista a nova instrução, ou ela ainda não esteja configurada, a interface informa a CPU a inexistência da instrução. No modo de reconfiguração, a interface funciona como um bloqueador e nega a execução de novas instruções. A *Figura 14* abaixo mostra os sinais envolvidos na operação deste módulo.

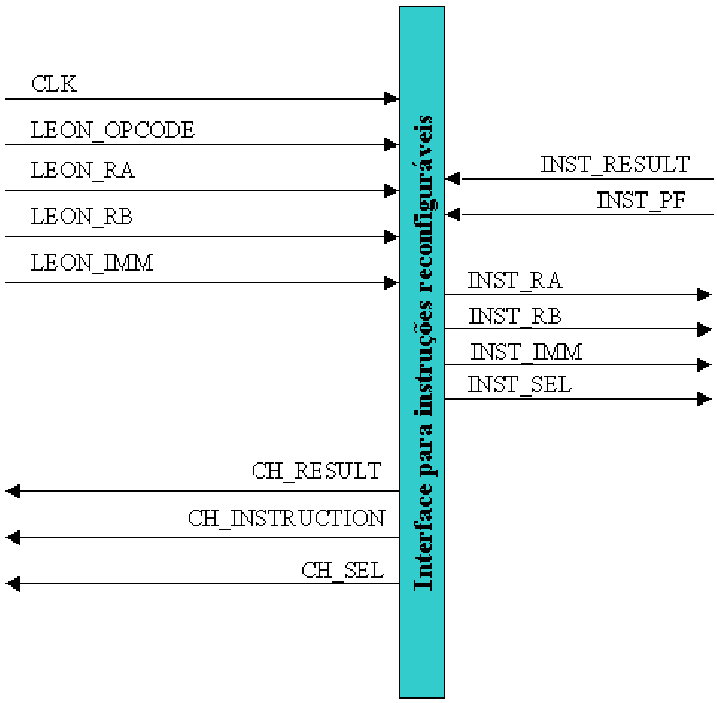


Figura 14 – Modulo de interface para novas instruções

O funcionamento deste módulo, será mostrado através do caminho de dados envolvido na execução das instruções.

Partido do suposto que já exista uma instrução no módulo reconfigurável, pode-se chegar aos seguintes cenários:

Cenário 1

Neste cenário será mostrado o processamento de uma instrução pertencente ao conjunto de instruções da CPU. Assim, no estágio de Busca, a CPU obtém a instrução a ser decodificada. No estágio de decodificação, os dados são passados a interface, que então verifica se o *opcode* recebido consta entre o conjunto de instruções da interface e se esta

instrução se encontra presente, no módulo de instrução. Como se trata de uma instrução nativa da CPU, a interface reconhece como uma instrução ilegal e envia esta informação para a CPU, através do sinal *CH_INSTRUCTION*, com nível lógico '0'. Neste caso a CPU reconhece a instrução e sua execução persegue normalmente.

Cenário 2

Neste próximo cenário, a instrução a ser executada pela CPU não está relacionada em seu conjunto de instruções. Assim, no estágio de decodificação, a instrução será considerada inválida *(ILLEGAL_INST: = '1')*. De forma semelhante ao cenário anterior, as informações da instrução a ser executada são passadas à interface, e é verificado que a instrução recebida pertence ao conjunto de instruções desta interface.

Falta verificar se a instrução reconhecida se encontra presente no módulo de instrução. O sinal de entrada *INST_PF* é responsável por esta verificação. Assim, há duas alternativas possíveis: a instrução se encontra presente no módulo de instrução ou não.

Cenário 2.1

Caso a instrução se encontre presente no módulo de instrução, e sabendo que se trata de uma instrução válida da interface, há a necessidade de anular o sinal *"ILLEGAL_INST"*, *setado* pela unidade de inteiros do processador. Este sinal indica que a CPU não reconheceu a instrução como sendo uma instrução válida. Assim, através da seguinte atribuição: *"ILLEGAL_INST:=ILLEGAL_INST* **AND NOT** *CH_INSTRUCTION"*, garantimos que o sinal de instrução inválida receberá nível lógico '0', visto que *CH_INSTRUCTION"* possui nível lógico '1'. Em seguida, já no estágio de execução, a instrução que se encontra do módulo reconfigurável é efetivamente executada e o resultado transferido de volta a CPU através da interface. Os sinais *INST_RESULT* e *CH_RESULT* são responsáveis por repassar esta informação, como pode ser visto na *Figura 14* acima. Além destes sinais também é passado a CPU, o sinal *CH_SEL*, responsável por selecionar a saída da ALU de acordo com a operação realizada. Nesta etapa, a saída da ULA receberá o

valor de *"CH_RESULT"*, através da atribuição a seguir: *("ALU_RESULT:= CH_RESULT")*. Isto ocorre quando a instrução executada, trata-se de uma nova instrução.

Cenário 2.2

Caso a instrução não se encontre presente no módulo de instrução, o sinal *INST_PF* retorna nível lógico '0' e mesmo ocorre com *CH_INSTRUCTION*. Desta forma *ILLEGAL_INST* continuará *setado ("ILLEGAL_INST:=ILLEGAL_INST* **AND NOT** *CH_INSTRUCTION")*, e a instrução será invalidada.

Os cenários ilustrados acima, mostraram os principais fluxos de dados que pode ocorrer no funcionamento da arquitetura, mostrando o significado dos sinais envolvidos e a detalhes da comunicação entre os módulos.

6.3 MÓDULO DE INSTRUÇÃO

Este módulo é responsável por conter a operação da instrução a ser executada através da interface definida. Como dito no início do capítulo, este módulo é alocado em uma área reconfigurável da FPGA, podendo se dinamicamente alterado. Uma característica importante deste módulo é seu tempo de execução, que é de um ciclo, semelhante às instruções implementadas no Leon. Assim tudo se passa como se a nova instrução fizesse parte da própria CPU. Este módulo é mostrado na figura abaixo.

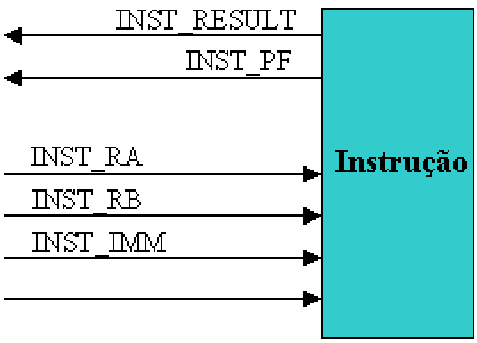


Figura 15 – Modulo de Instrução

6.4 ARQUITETURA FINAL DO SISTEMA

Agora, depois que todos os módulos já foram definidos, apresentamos o modelo da arquitetura final proposta, na figura abaixo. Neste diagrama, foi adicionado um módulo responsável por interligar a área fixa e a área reconfigurável, a fim de permitir a comunicação entre os módulos, chamado de *Bus Macro*. Este módulo, como já relatado no capítulo 4 deste trabalho, não sofre modificações durante as mudanças de configurações, através da utilização de canais de roteamento fixo.

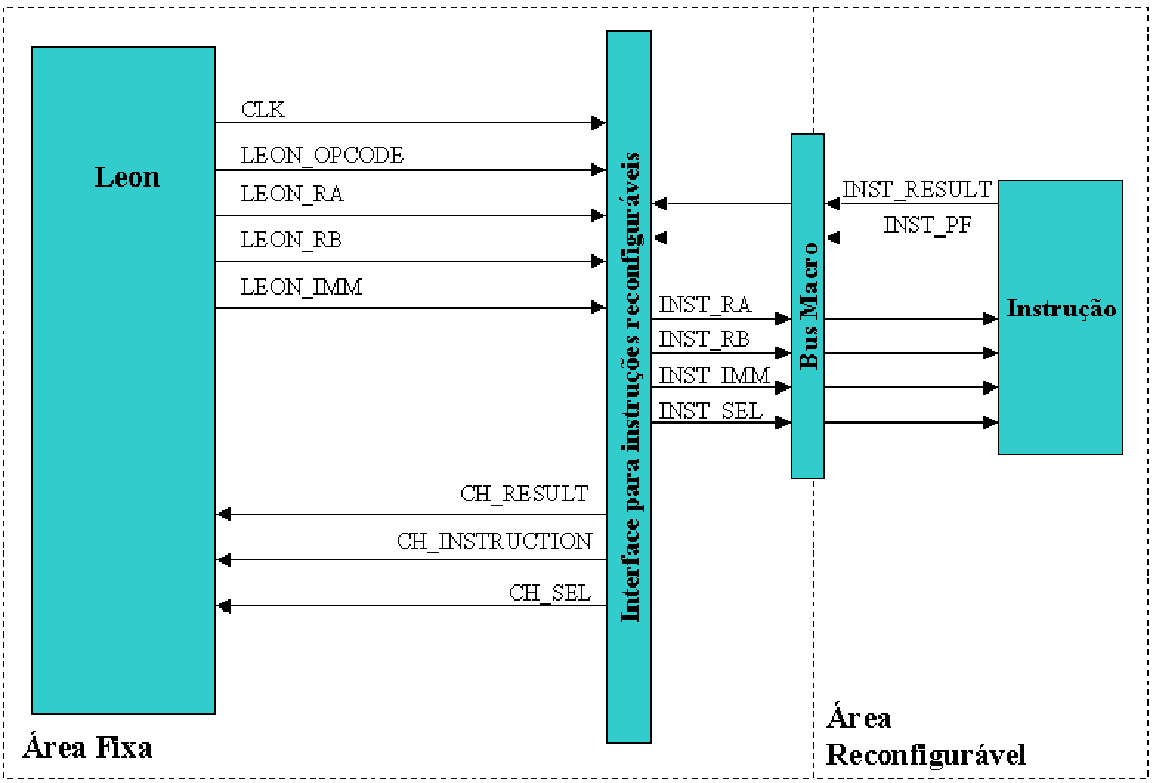


Figura 16 – Arquitetura Final proposta

6.5 ESTUDO DE CASO

Nesta seção será apresentado um estudo de caso para o processo de reconfiguração dinâmica utilizando a plataforma definida na seção anterior. Neste estudo, a CPU Leon irá executar um programa fictício representado pelas seguintes instruções: ADD, SUB, NEW, RECONFINST, SUB, RECONFINST, NEW, ADD. A instrução NEW, especificada neste programa, é uma instrução especial dinamicamente reconfigurável. Já a instrução RECONFINST é responsável por bloquear e desbloquear a interface, configurando o modo de operação desta.

Inicialmente, o módulo de instrução reconfigurável, possui a instrução especial "OR". Desta forma, à medida que a CPU encontra uma instrução NEW, na configuração atual, uma operação "OR" será executada. O cenário inicial, para a execução do programa é apresentado abaixo. Note que há um dispositivo externo, responsável pela reconfiguração do módulo de instrução, representado em tom cinza claro. Este módulo se encontra desabilitado no momento.

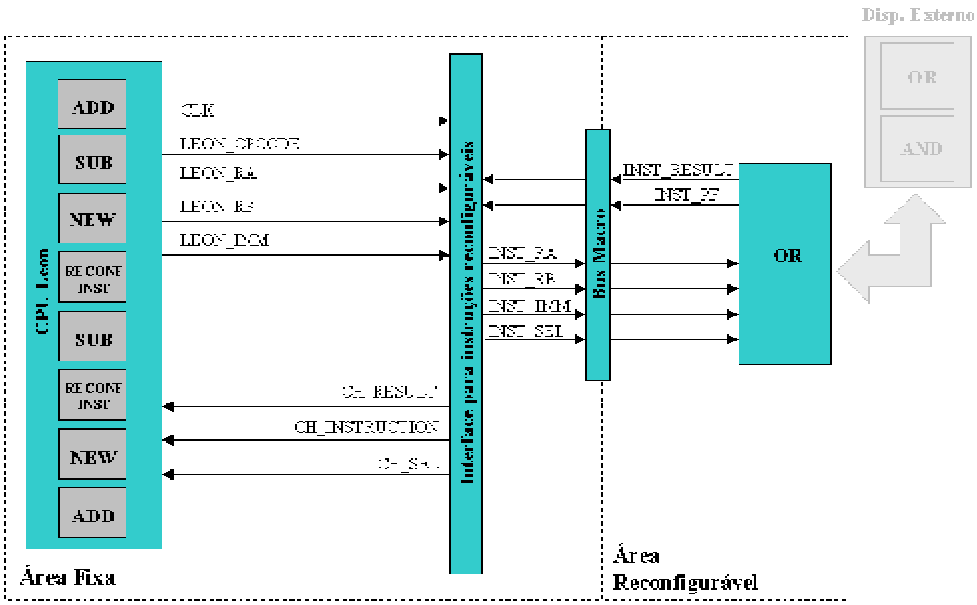


Figura 17 – Cenário inicial do estudo de caso

A figura seguinte mostra o início da execução da instrução de ADD, primeira instrução do programa.

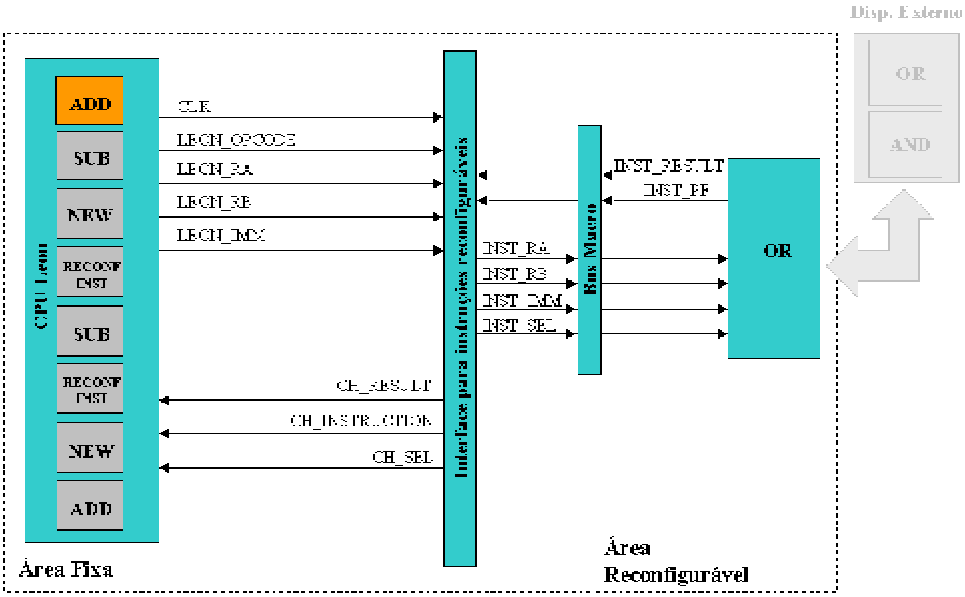


Figura 18 – Início da execução da instrução de ADD

Em seguida, a CPU envia os dados recebidos durante o estágio de decodificação para a interface, que então retorna, informando que o *opcode* recebido não pertence a seu conjunto de instruções válidas. Nesta situação, a CPU executa normalmente a instrução decodificada e inicia uma nova busca de instrução. As duas próximas figuras mostram este comportamento.

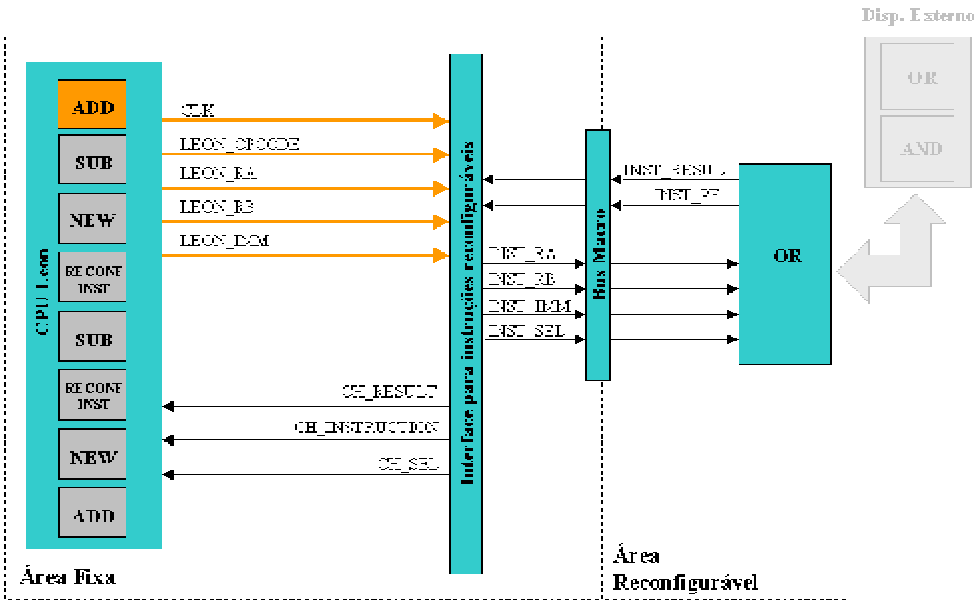


Figura 19 – Execução da instrução de ADD (envio das informações)

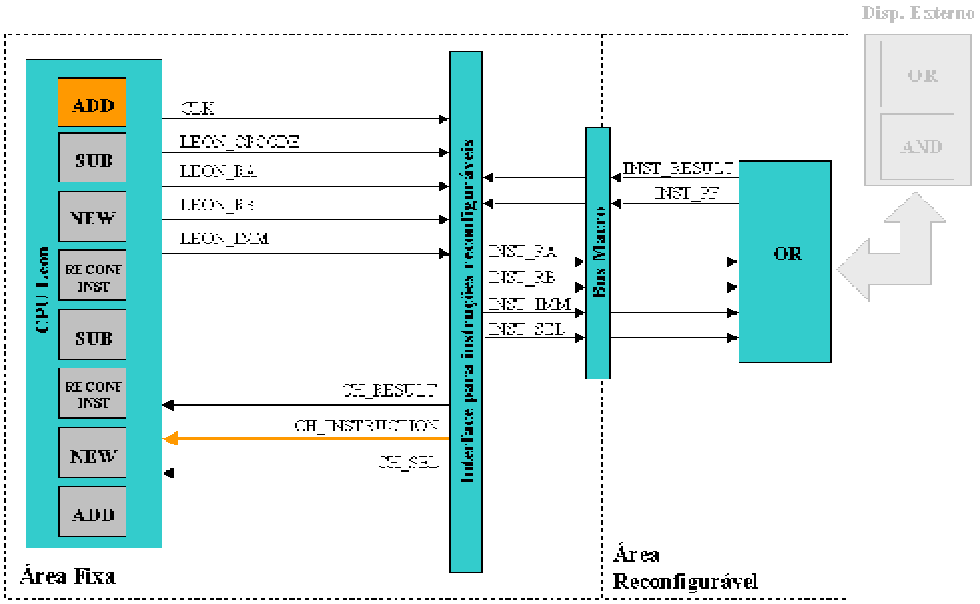


Figura 20 - Execução da instrução de ADD (retorno da informação)

Nesta etapa do programa, a instrução especial “OR” é executada pela CPU. Note que além desta instrução ter que pertencer ao conjunto de instruções da interface, o *flag* de instrução presente (INST_PF), fornecido pelo módulo de instrução, também deve está *setado*.

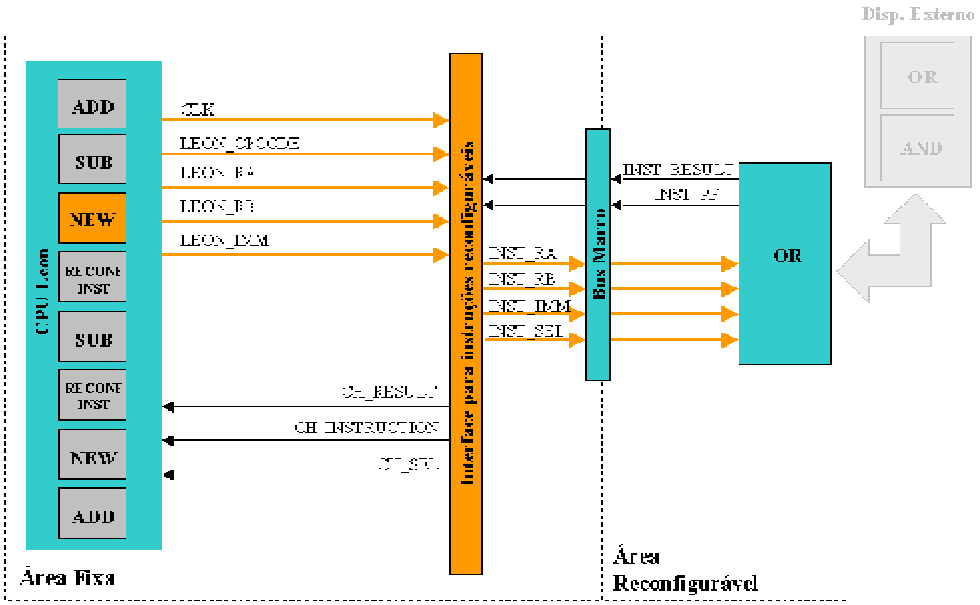


Figura 21 – Execução da instrução especial OR

Após receber as informações da CPU, o módulo reconfigurável processa a instrução e o retorna o resultado à interface, que então repassa para a *CPU*. Isto pode ser visto na figura abaixo.

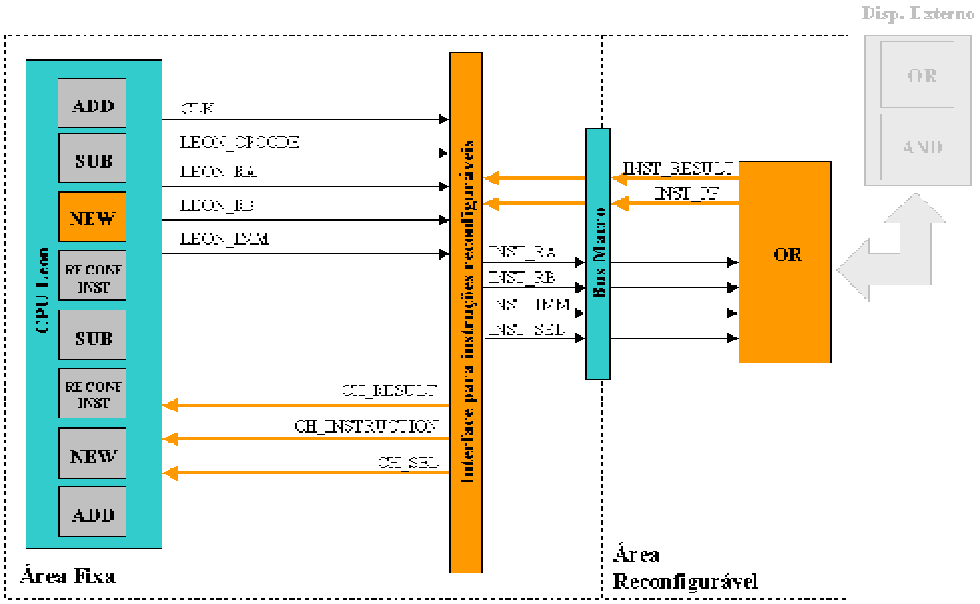


Figura 22 – Resultado da instrução especial OR

A próxima instrução a ser executada é a instrução RECONFINST. Ao final de sua execução a interface entra no modo de reconfiguração, ficando desta forma bloqueada. Neste modo de operação, a instrução localizada no módulo reconfigurável, fica impedida de ser executada, evitando que esta instrução seja acessada durante o processo de reconfiguração. Até que outra instrução RECONFINST seja novamente executada, a interface permanece bloqueada. Entretanto, a CPU continua seu processamento normal, executando a instrução de SUB. É obvio que o processo de reconfiguração é muito mais lento que a execução de uma instrução de SUB, mas para efeito ilustrativo, estamos considerando este fato aqui. Em uma aplicação real, várias instruções seriam executadas durante uma reconfiguração dinâmica. Isto pode ser mais claramente entendido nas figuras abaixo.

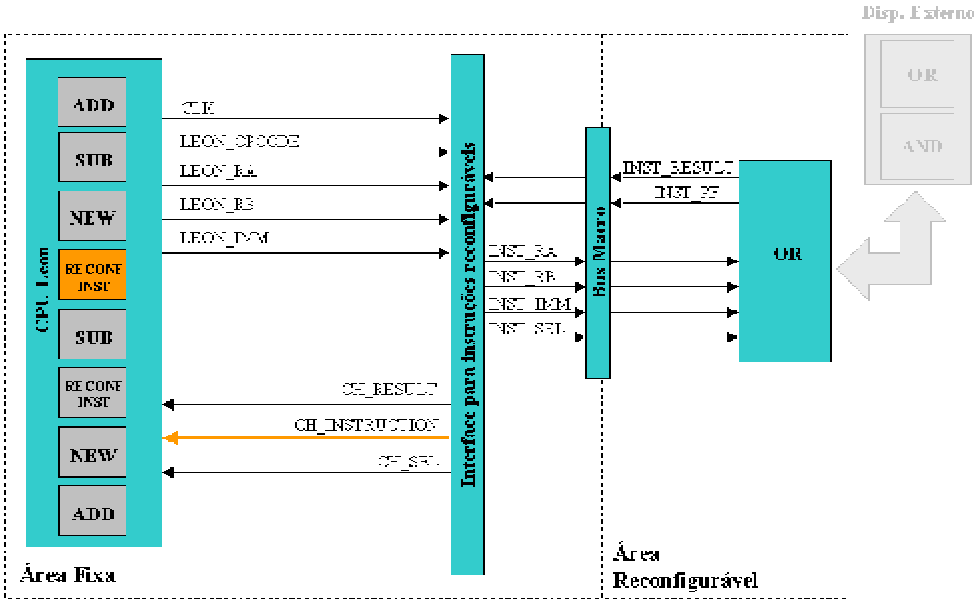


Figura 23 – Execução da instrução RECONFINST (modo de reconfiguração)

Neste momento, o módulo de instrução está sendo reconfigurado. A instrução “OR” localizada neste módulo, será substituída pela instrução “AND”, que se encontra no dispositivo externo. Note que a interface se encontra bloqueada.

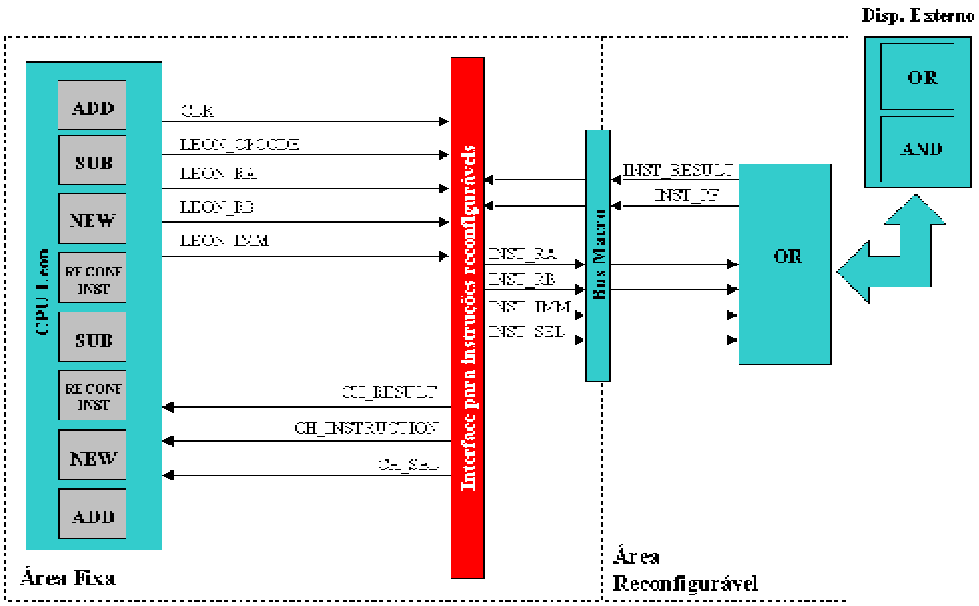


Figura 24 – Interface em modo de reconfiguração

Em seguida, logo após a execução da instrução de SUB, a instrução RECONFINST é executada novamente, configurando a interface para o modo ativo

novamente. Neste modo, instruções localizadas no módulo reconfigurável podem ser executadas.

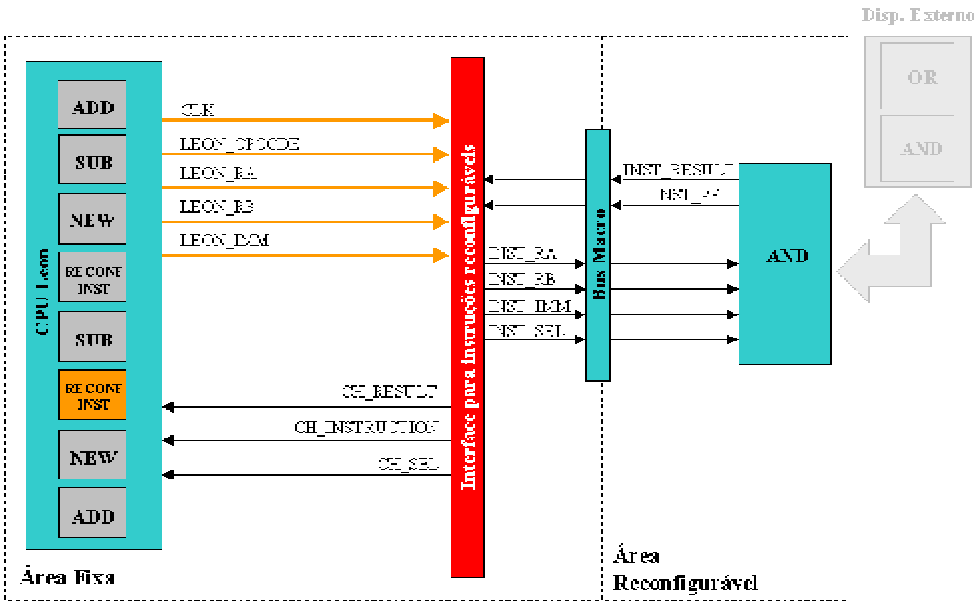


Figura 25 - Execução da instrução RECONFINST (modo ativo)

Assim, quando a instrução NEW for novamente executada, a CPU irá receber o resultado de uma operação de “AND” e não de “OR”, como inicialmente. Isto é mostrado abaixo.

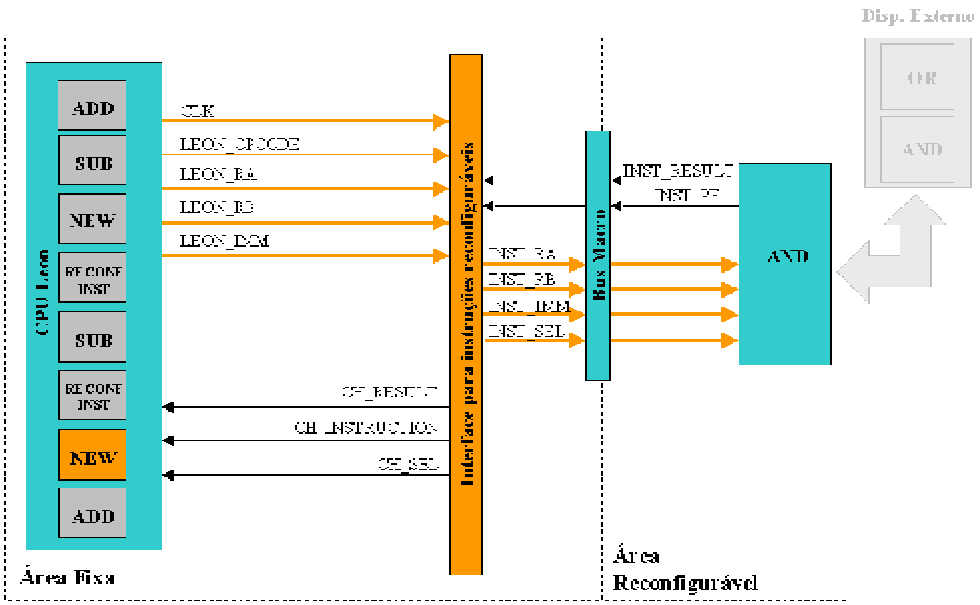


Figura 26 – Execução da instrução AND (envio das informações)

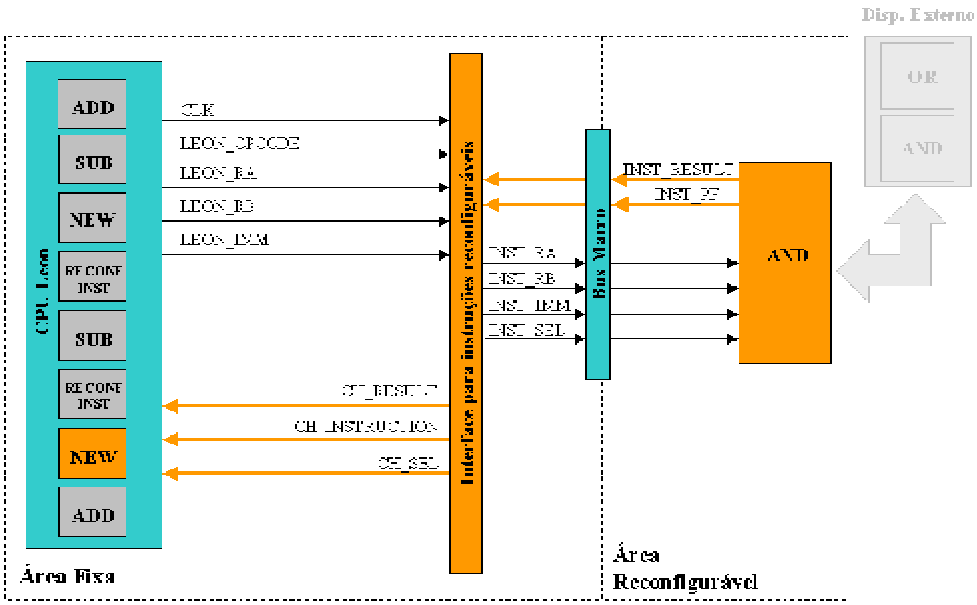


Figura 27 - Execução da instrução AND (retorno do resultado)

7 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho foi proposto o estudo de uma arquitetura dinamicamente reconfigurável para prototipação rápida de sistemas digitais baseado na CPU Leon. Assim, estabelecida a plataforma, foi estudado a CPU Leon e seus módulos, os dispositivos lógicos programáveis e o processo de reconfiguração dinâmica destes dispositivos. Em seguida, foi proposta uma arquitetura para a reconfiguração estática e dinâmica de instruções para a plataforma definida, onde formam especificados os módulos da interface de instruções e o módulo para instruções dinamicamente reconfiguráveis, levando em consideração as instruções tipo R, definidas no padrão SPARC [SPARC].

Concluímos que é possível aplicar os conceitos de reconfigurabilidade dinâmica de instruções em inúmeras áreas de aplicação. Podemos citar, a adaptação do hardware na realização de algoritmos específicos, aplicações em computação móvel, onde os dispositivos possuem recursos limitados e necessidade de baixo consumo de potência, como também, aplicações em sistemas críticos, onde a disponibilidade é fundamental. Entretanto, ainda há limitações para o desenvolvimento de um bom projeto reconfigurável, visto que grande parte das ferramentas CAD, disponíveis no mercado, ainda não fornecem todos os recursos necessários para isso.

Entre as propostas de trabalhos futuros, temos a implementação da arquitetura proposta, na plataforma alvo, como a maior prioridade. Em seguida pode-se pensar na extensão desta arquitetura a fim de tornar possível a reconfiguração de mais de um módulo de instrução, assim como a extensão dos tipos de instruções possíveis, possibilitando a ampliação de seu campo de aplicação deste trabalho.

APÊNDICES

APÊNDICE A - DESCRIÇÃO EM VHDL DO MÓDULO DA INTERFACE

```
-------------------------------------------------------------------------------------------------------
-- Entity: Chameleon Interface for Reconfigurable Instructions
-- File:    ch_reconf_interf.vhd
-- Authors: Julio Alexandrino de Oliveira Filho - GRECO - CIN - UFPE
--              Luciana Shiroma Montari - LSC - IC - UNICAMP
--              Rodolfo Jardim de Azevedo - LSC - IC - UNICAMP
-- Description: Implements the ChameLeon Interface Unit for Reconfigurable Instructions
-------------------------------------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned."+";
use IEEE.std_logic_unsigned."-";
use IEEE.std_logic_unsigned.conv_integer;
use work.target.all;
use work.config.all;
use work.sparcv8.all;
use work.iface.all;
use work.macro.all;
use work.tech_map.all;
use work.multlib.all;

entity ch_reconf_interf is

  port (
-- Input signals from Leon
    clk          : in clk_type;                                -- Clock signal
    leon_opcode    : in std_logic_vector(7 downto 0);-- Instruction op_code from Leon
    leon_ra        : in std_logic_vector(31 downto 0);     -- First input (register) from Leon
    leon_rb        : in std_logic_vector(31 downto 0);  -- Second input (register)from Leon
    leon_imm           : in std_logic_vector(7 downto 0);            -- Third input (imediate) from Leon
-- Input signals from Instruction Module
    inst_result    : in std_logic_vector(31 downto 0);  -- Instruction result vector
    inst_pf        : in std_logic_vector(1 downto 0);        -- Instruction presence flag,
                                        -- used for identification
-- Output signals to Leon
    ch_result      : out std_logic_vector(31 downto 0);-- ChameLeon result
    ch_instruction : out std_logic;                          -- ChameLeon valid instruction flag
    ch_sel             : out std_logic;                          -- Selection bit
-- Output signals to Instruction Module
    inst_ra        : out std_logic_vector(31 downto 0); -- First instruction parameter
    inst_rb        : out std_logic_vector(31 downto 0); -- Second instruction parameter
    inst_imm       : out std_logic_vector(7 downto 0);-- Third instruction parameter
    inst_sel       : out std_logic_vector(1 downto 0)); -- Inst. selection line
end ch_reconf_interf;
```

```
architecture rtl of ch_reconf_interf is

-- To be defined here or in SPARCV8.VHD
constant CH_OPCODE1     : std_logic_vector(7 downto 0) := "10001001";

-- First Chameleon opCode slot
constant RECONFINST     : std_logic_vector(7 downto 0) := "10001101";

-- Second Chameleon opCode slot
constant CH_OPCODE2     : std_logic_vector(7 downto 0) := "10011001";
constant STANDARD_PARAM : std_logic_vector(31 downto 0):= (others => '0');
constant STANDARD_PARAM_IMM : std_logic_vector(7 downto 0) := (others => '0');
constant STANDARD_RESULT : std_logic_vector(31 downto 0) := (others => '0');

-- Active Instructions Flag (1 = Active Inst,  0 = Reconf. Mode)
signal ch_active_instruction     : std_logic;

-- Decode stage active instruction flags
signal ch_active_instruction_in  : std_logic;
signal inst_sel_in               : std_logic_vector(1 downto 0);
signal select_ch_result      : std_logic;          -- Instruction(1) or standard(0) result
signal select_ch_result_in : std_logic;

begin  -- rtl

-- Interface Decoder stage
-- This module is responsible for controling Chameleon instruction validation (it must be a valid op_code and
--  it must be an implemented instruction).  It also organizes signalization for Leon and for instruction result
--  selection structures.

  interf_decoder: process(leon_opcode,inst_pf, ch_active_instruction)

  variable interfmode          : std_logic;          -- Select interface mode (active or reconfiguration)
  variable select_result       : std_logic;          -- Select result source
  variable inst_selection      : std_logic_vector(1 downto 0); -- Selection lines
  variable invalid_instruction  : std_logic;         -- Identifies this instruction as an Chameleon instruction
  variable sel_ch_instruction   : std_logic;         -- Selects Chameleon Result on Leon

  begin

-- Interface Mode evaluation
    interfmode := '1';
--     interfmode := ch_active_instruction;           -- Reconfiguration Mode is Default
--     case leon_opcode is
--     when RECONFINST =>
--          interfmode := not interfmode;                  -- RECONFINST toggles interface mode between
--     when others => null;                        -- active and reconfiguring modes.
--     end case;

    ch_active_instruction_in <= interfmode;

-- Invalid instruction evaluation
    invalid_instruction := '1';                 -- Invalid Instruction is Default
    case leon_opcode is
    when RECONFINST =>
      invalid_instruction := '0';               -- It is always a valid chameleon instruction
```

```
    when CH_OPCODE1 =>
      if ((ch_active_instruction and inst_pf(0)) = '1') then -- All Chameleon instructions are valid only if
        invalid_instruction := '0';              -- interface is on active mode and if instruction
      end if;                                 -- is really present.
    when CH_OPCODE2 =>
      if ((ch_active_instruction and inst_pf(1)) = '1') then
        invalid_instruction := '0';
      end if;
    when others =>
        invalid_instruction := '1';              -- Even Leon instructions are classified as
    end case;                                  -- invalid for Chameleon Interface

    ch_instruction <= not invalid_instruction;

-- Decoding instructions
    select_result := '0';                      -- Default result
    inst_selection := (others => '0') ;        -- Selects no instructions
    sel_ch_instruction  := '0';                -- Selected result is from Leon
    if (ch_active_instruction = '1') then
      case leon_opcode is
      when RECONFINST =>
        select_result := 0'0';                 -- RECONFINST selects chameleon default result
        select_result := '1';                  -- Chameleon result
        sel_ch_instruction  := '1';            -- Selected result is from ChameLeon Interface
      when CH_OPCODE1 =>
        if (inst_pf(0) = '1') then
          inst_selection(0) := '1';            -- Select ChameLeon instruction 0
          select_result := '1';                -- Chameleon result
          sel_ch_instruction  := '1';          -- Selected result is from ChameLeon Interface
        end if;
      when CH_OPCODE2 =>
        if (inst_pf(1) = '1') then
          inst_selection(1) := '1';            -- Select ChameLeon instruction 1
          select_result := '1';                -- Chameleon result
          sel_ch_instruction  := '1';          -- Selected result is from ChameLeon Interface
        end if;
      when others => null;                     -- No instruction were found
      end case;
    end if;

    select_ch_result_in <= select_result;
    ch_sel <= sel_ch_instruction;
    inst_sel_in <= inst_selection;

  end process;


-- Interface Execution stage
-- These modules are active when the instruction goes to Leon execution estage on the pipeline.
-- They are all synchronized by the clock

-- Selection for result

  result_mux: process (select_ch_result,inst_result)
  begin
    case (select_ch_result) is                 -- Selects either the STANDARD_RESULT
```

```
    when '0' =>                                 -- (when no instruction is performed) or
      ch_result <= STANDARD_RESULT;             -- the instruction result is passed to Leon
    when others =>                              -- pipeline.
      ch_result <= inst_result;
    end case;
  end process;

-- Selection for parameters
  param_mux: process (ch_active_instruction, leon_ra, leon_rb, leon_imm)
  begin
    case (ch_active_instruction) is
    when '1' =>
      inst_ra <= leon_ra;                       -- Passes for instruction the params sent by
      inst_rb <= leon_rb;                       -- Leon or the standard_param, in case of
      inst_imm <= leon_imm;                     -- reconfiguration mode.
    when others =>
      inst_ra <= STANDARD_PARAM;
      inst_rb <= STANDARD_PARAM;
      inst_imm <= STANDARD_PARAM_IMM;
    end case;
  end process;

-- Clocked stage between Interface Decoder and Execution stage

  interf_registers:process (clk)
  begin
   if (rising_edge(clk)) then
      select_ch_result <= select_ch_result_in;      -- Selects decoded result
      ch_active_instruction <= ch_active_instruction_in;-- Selects Interface mode
      inst_sel <= inst_sel_in;                  -- Selects correct instruction
   end if;
  end process;

end rtl;
```

APÊNDICE B -DESCRIÇÃO EM VHDL DO MÓDULO DE INSTRUÇÕES

```
--------------------------------------------------------------------------------
-- Entity: chameleon
-- File:    chameleon.vhd
-- Author:          Rodolfo Jardim de Azevedo - LSC - IC - UNICAMP
-- Description:     Implements the ChameLeon reconfigurable unit
--------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned."+";
use IEEE.std_logic_unsigned."-";
use IEEE.std_logic_unsigned.conv_integer;
use work.target.all;
use work.config.all;
use work.sparcv8.all;
use work.iface.all;
use work.macro.all;
use work.tech_map.all;
use work.multlib.all;

entity ChameLeon is

  port (
    ra           : in  std_logic_vector(31 downto 0);   -- First input (register)
    rb           : in  std_logic_vector(31 downto 0);   -- Second input (register)
    imm      : in  std_logic_vector(7 downto 0);    -- Third input (imediate)
    sel        : in  std_logic_vector(1 downto 0);    -- ChameLeon enable
    inst_pf   : out std_logic_vector (1 downto 0);   -- Chameleon instruction selection
    result    : out std_logic_vector(31 downto 0));  -- ChameLeon output

end ChameLeon;

architecture rtl of ChameLeon is

signal instruction1_result : std_logic_vector(31 downto 0);
signal instruction2_result : std_logic_vector(31 downto 0);
signal selected_result     : std_logic_vector(31 downto 0);

begin  -- rtl

  inst_pf <= "11";
  mux : process (ra,rb,sel,selected_result)
  begin
    case sel is
    when "01" =>
      result <= selected_result;
    when "10" =>
      result <= ra and rb;
    when others =>
      result <= (others => 'Z');
    end case;
  end process;
```

```
  instruction1: process (ra, imm)
  begin
    if (ra > "00000000000000000000000001011000") then
      instruction1_result <= "00000000000000000000000001011000";
    else
      instruction1_result <= ra;
    end if;

    if (ra(31) = '1') then
      instruction2_result <= (others => '0');
    else
      instruction2_result <= ra;
    end if;
  end process;

  res_mux: process (instruction1_result, instruction2_result, imm)
  begin
   case imm is
    when "00000000" =>
      selected_result <= instruction1_result;
    when "00000001" =>
      selected_result <= instruction2_result;
    when others =>
      selected_result <= (others => '0');
    end case;
  end process;

end rtl;
```

APÊNDICE C – FERRAMENTA DE CONFIGURAÇÃO DO LEON

A versão da CPU Leon utilizada neste trabalho foi a *VHDL 2-1.0.21*, disponibilizada em: http://www.gaisler.com

Para iniciar a ferramenta de configuração, digite, no diretório raiz do Leon, o comando: ***"make wconfig"***. Uma janela igual à mostrada abaixo deverá ser exibida.

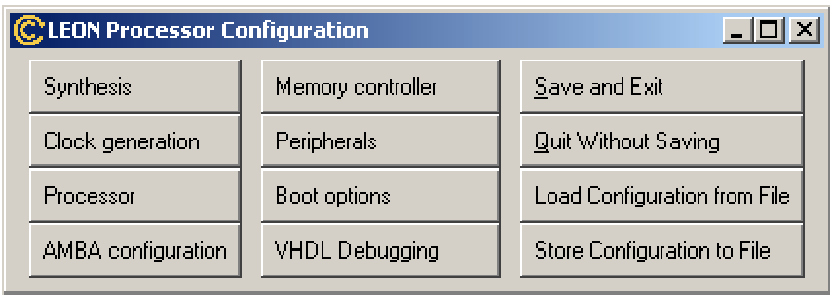


Figura 28 - Janela principal da Ferramenta de configuração do Leon

Através desta ferramenta, poderemos definir um processador totalmente customizado, habilitando e desabilitando módulos e modificando as configurações dos dispositivos habilitados. O processo que será descrito em seguida, refletirá a configuração-base, que será adotada durante o trabalho. Primeiramente, será configurado o processo de síntese do Leon. Para isso, na janela principal da ferramenta de configuração, clique no botão *Synthesis*. Uma janela, igual à mostrada na *Figura 29* será aberta. Escolha a tecnologia alvo, neste caso, Xilinx Virtex II.

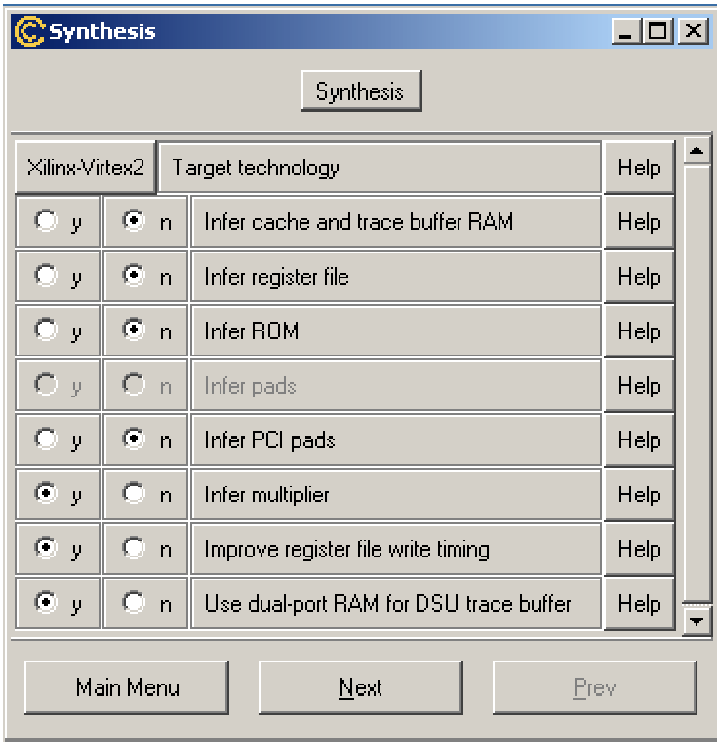


Figura 29 - Configuração da Síntese

O próximo passo é configurar os componentes do processador. Para isso, na janela principal, clique no botão *Processor*. Trabalharemos com a unidade de ponto flutuante e co-processador desabilitados. Certifique-se que estas estão mesmo desabilitadas. A figura abaixo mostra a janela de configuração do processador a que nos referimos.

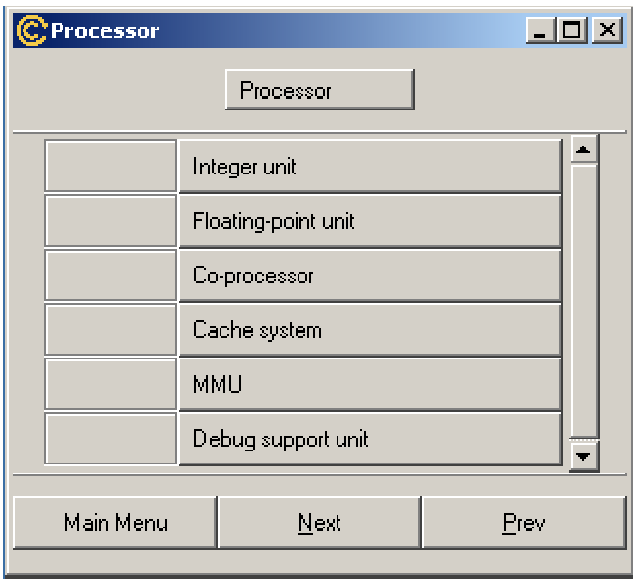


Figura 30 – Configurações do Processador

Para configurar a unidade de Inteiros, clique em *Integer Unit*, na janela *Processor*. Verifique se *fast jump-adress*, *ICC interlock* e *fast instruction deconding* estão habilitados. Caso não estejam, habilite-os.

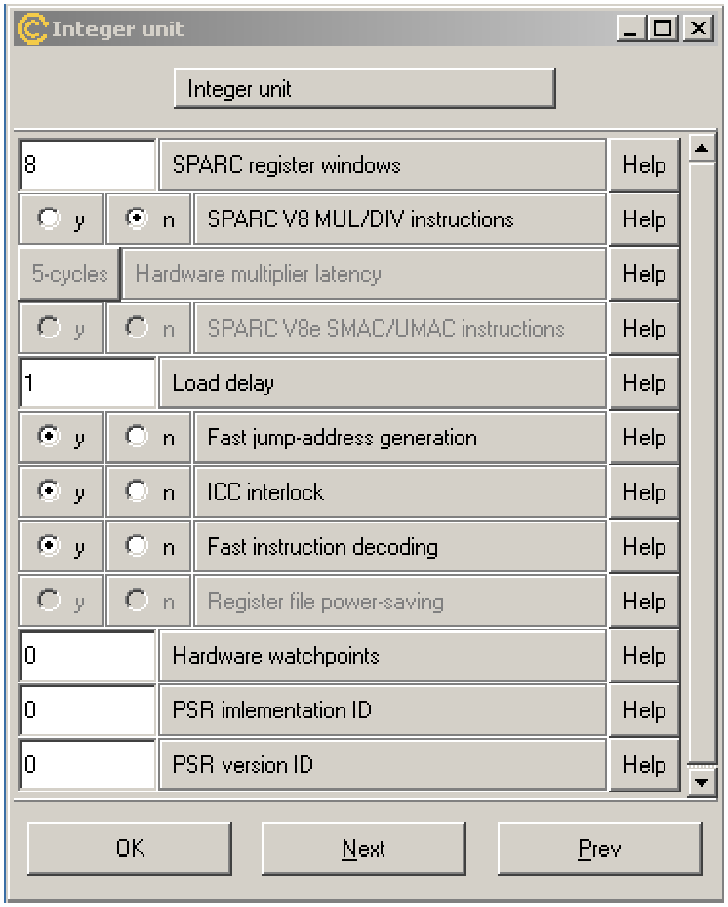


Figura 31 – Configuração da Unidade de Inteiros

Ainda na janela de configuração do processador, clique em *Cache System*, para configurar as caches de instrução e de dados. Estas configurações podem ser modificadas de acordo como a quantidade de RAM disponíveis na FPGA. Na nossa configuração, utilizaremos associatividade de dois conjuntos, como 1kbyte por conjunto e 32 bytes por linha. Como algoritmo de substituição de cache utilizaremos o randômico.

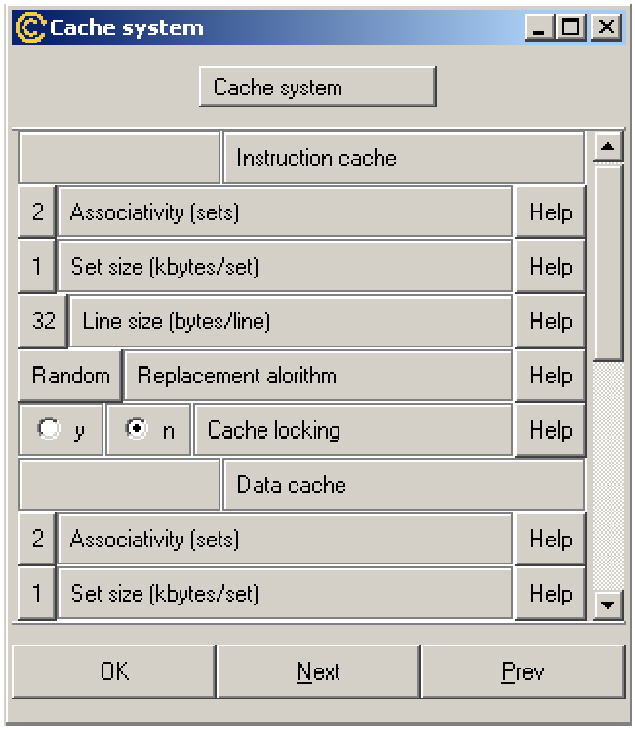


Figura 32 – Configuração das memórias caches

Habilitaremos a unidade de suporte à depuração. Isso é feito na janela *Debug suport unit*, mostrada abaixo.

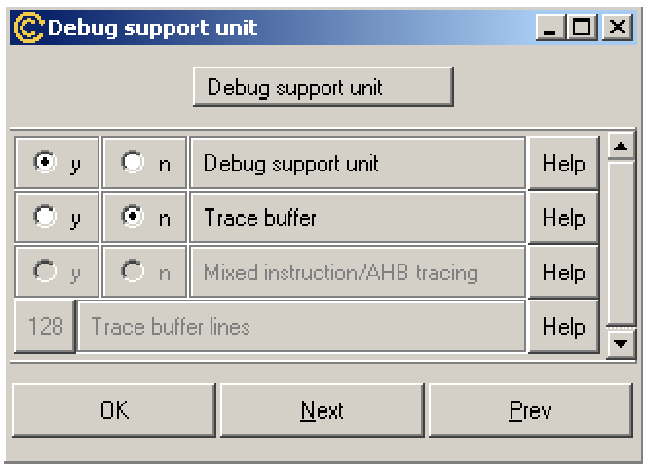


Figura 33 – Configuração da Unidade de Depuração

Na janela de configuração do controlador de memória, isto é, *Memory controller*, desabilite todas as opções visto que será utilizada a própria RAM do processador. Isto será configurado na janela de configuração dos periféricos.

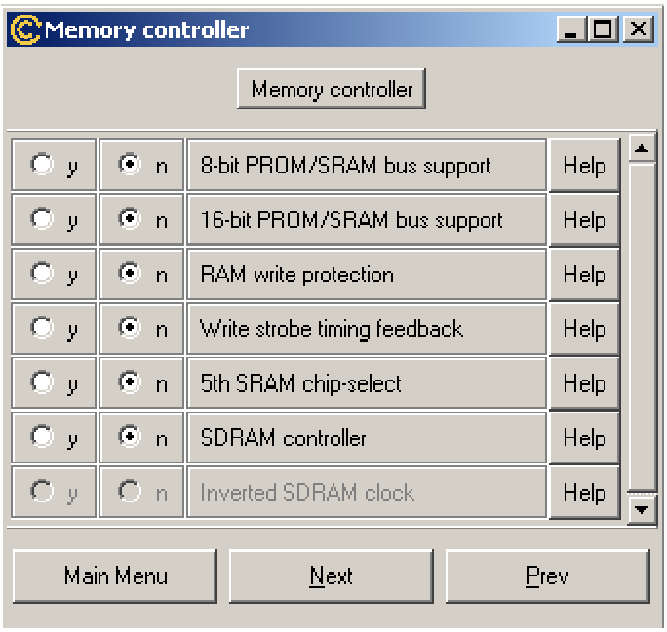


Figura 34 – Configuração do controlador de memória

Na janela de configuração dos periféricos, habilite Leon *configuration register* e *on-chip AHB RAM* e defina o tamanho da RAM para 8kbytes. Como não utilizaremos a interface Ethernet e a PCI, desabilite-as.

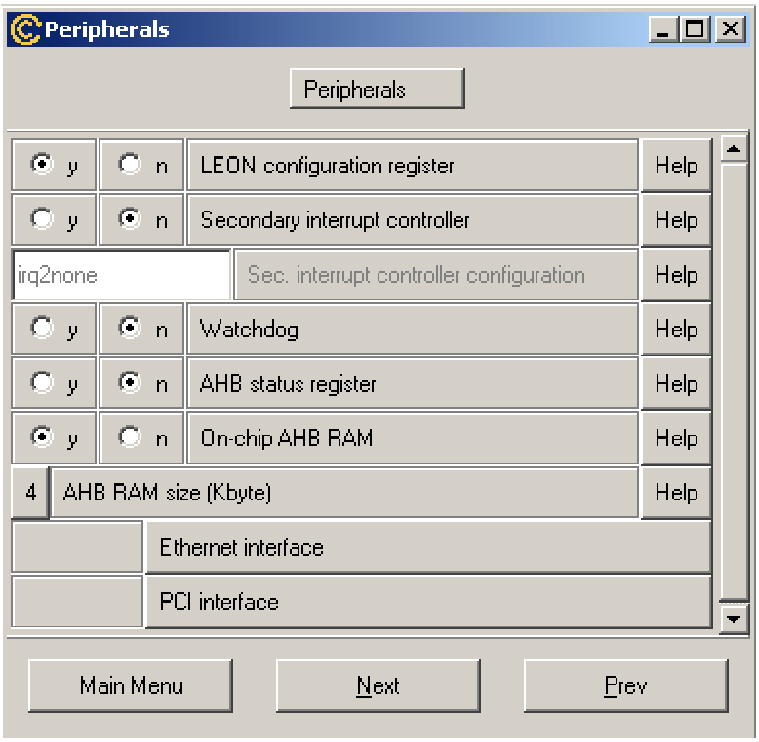


Figura 35 - Configuração dos periféricos

Em *Boot options*, na configuração do item *Boot selection*, escolha Internal-Prom. Desta maneira, o boot será dado por Prom interna. A taxa de transmissão da UART foi definida em 38400Kbs.

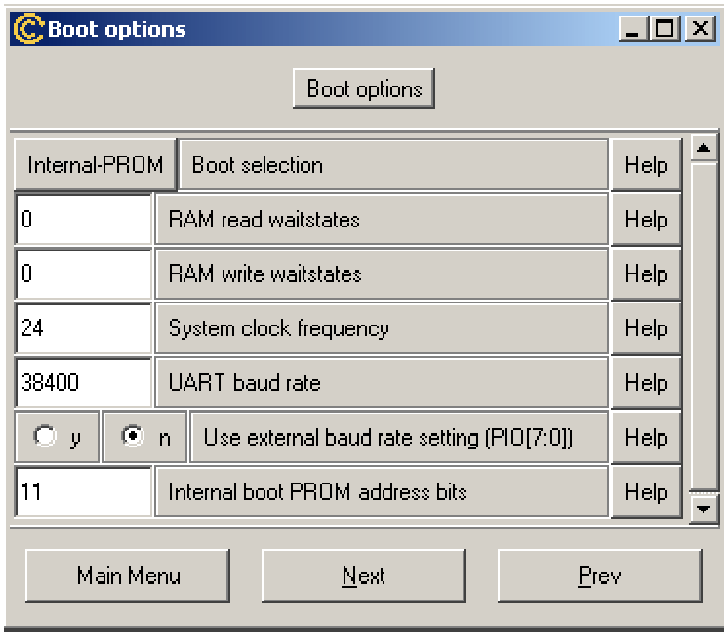


Figura 36 – Configuração de Boot

Finalmente, a configuração do processador está finalizada. Na janela principal, pressione o botão *save and exit*, isto irá salvar as configurações que acabamos de fazer. Agora, para concluir, processo, precisamos gerar os arquivos VHDL de configuração através do comando ***make dep*** no prompt de comandos.

REFERÊNCIAS BIBLIOGRÁFICAS

[Knapp; Tavana] Knapp,Steve; Tavana,Danesh. Field Configurable System-on-Chip Device Architecture. Triscente Corporation 301 N. Whisman Road Mountain View.

[Ribeiro; Marques] Ribeiro, Alexandre Alves de Lima; Marques, Eduardo. Reconfiguração Dinâmica e Remota de FPGAs.12f. Instituto de Ciências Matemática e de Computação, Universidade de São Paulo, São Paulo.

[Bray] Bray, Neil. Designing for the IP supermarket. ARM Limited. Cambridge. United Kingdom.

[Küçükçakar] Küçükçakar, Kayhan. Analysis of Emerging Core-Based Design Lifecycle. Escalade Corp. Santa Clara.

[Junchao; Weiliang; Shaojun] Junchao,ZHAO;Weiliang, CHEN;Shaojun, Wei. Parameterized IP Core Design.Institute of Microeletronics. Tsinghua University. Beijing. China.

[Gaisler Research] Gaisler Research.*The LEON-2 Processor User's Manual XST Edition*.v.1.0.21.Göteborg,out.2003.90p. Disponível em:<http://www.gaisler.com/doc/leon2-1.0.21-xst.pdf>. Acesso em: 26 dez.2003.

[Menec Design] Menec Design. Virtex – II V2MB1000 Development Board User's Guide. Disponível em: <http://insight.legacy.memec.com/solutions/reference/xilinx/downloads/ V2MB_User_Guid_3_0.PDF>. Acesso em: 29 de dez.2003.

[The Leox Team] The Leox Team. *Free Resources for System on Chip*.Disponível em: <http://www.leox.org>. Acesso em: 26 dez.2003.

[ARM Limited] ARM LIMITED.*AMBA Specification*.maio.1999.Cap.1,p.13-26. Disponível em:<http://www.gaisler.com/doc/amba.pdf>.Acesso em: 26 dez.2003.

[Azevedo] Azevedo, Rodolfo. *O processador Leon*.Disponível em:http://www.dcc.unicamp.br/~rodolfo/mo801/> Acesso em: 26 de dez.2003.

[Baron] Baron, Max.*A GNU SPARC?.jan.2001.v.15.Cap.3*. Disponível em: <http://www.mdronline.com/mpr_public/editorials/edit15_03.html>. Acesso em :27 de dez.2003.

[Brown; Rose] Brown, Stephen; Rose, Jonathan. Architecture of FPGAs and CPLDs. 41f. Department of Electrical and Computer Engineering , University of Toronto.

[Clarke] Clarke, Peter.*European Space Agency launches free Sparc-like core* March 6, 2000. Disponível em: <http://www.eetimes.com/story/OEG20000306S0096>. Acessado em: 27 de dez.2003.

[Clarke, Peter] Clarke, Peter. *Momentum builds for open-source LONDON processors*. February 5, 2001. Disponível em: <http://www.eetimes.com/story/OEG20010201S0050>. Acessado em: 27 dez.2003.

[Pellerin;Douglas] Pellerin, David.;DOUGLAS,Taylor.*VHDL Made Easy*.New Jersey:Prentice Hall,1997.419p.

[Queiroz; Menotti] Queiroz, Daniel Cruz; Menotti,Ricardo; Bonato, Vanderlei. Síntese Lógica para FPGA. Departamento de Ciências da Computação e de Estatística, Universidade de São Paulo. 2003.

[Ribeiro] Ribeiro, Alexandre Alves de Lima; Marques, Eduardo. Reconfiguração Dinâmica e Remota de FPGAs.12f. Instituto de Ciências Matemática e de Computação, Universidade de São Paulo, São Paulo.

[SPARC] SPARC INTERNATIONAL INC.*The SPARC Architecture Manual*.v.8.Menlo Park,1992.Cap.2,p.11-17.Disponível em:<http://www.gaisler.com/doc/sparcv8.pdf>.Acesso em: 26 dez.2003.

[XILINX] XILINX.*ISE 6 In-Depth Tutorial*.1994-2003.130p. Disponível em:<http://www.xilinx.com>.Acesso em: 26 dez.2003.

[Zhang] ZHANG,Weijun.*VHDL Tutorial: Learn by Example.*Diponível em: <http://www.cs.ucr.edu/content/esd/labs/tutorial/>. Acesso em: 26 de dez.2003.

[Moraes, Mesquita] Moraes, Fernando; Mesquita, Daniel. Tendências em Reconfiguração Dinâmica de FPGAs. Faculdade de Informática PUCRC

[Boschetti] Boschetti, Marcos;Bampi, Sergio. Mecanismos de Reconfiguração Dinâmica Aplicados os Projeto de um Processador de Imagens Reconfigurável. Instituto de Informática.Universidade Federal de Rio Grande do Sul. Porto Alegre, Brasil.

[Mesquita] Mesquita, Daniel Gomes. Contribuições para reconfiguração parcial, remota e dinâmica de FPGAs . Pontifícia Universidade Católica do Rio Grande do Sul. Faculdade de Informática. Porto Alegre. Março, 2002.

_____

Halmos Fernando do Nascimento
Aluno

_____

Manoel Eusébio de Lima
Orientador